**Functions**

The **math** **functions**, calc(), min(), max(), and clamp() allow mathematical expressions with addition (+), subtraction (-), multiplication (\*), and division (/) to be used as component values

**calc() function**

The calc() CSS function lets you perform calculations when specifying CSS property values. It can be used with <length>, <frequency>, <angle>, <time>, <percentage>, <number>, or <integer> values.

Basic example:

.main-content {

/\* Subtract 80px from 100vh \*/

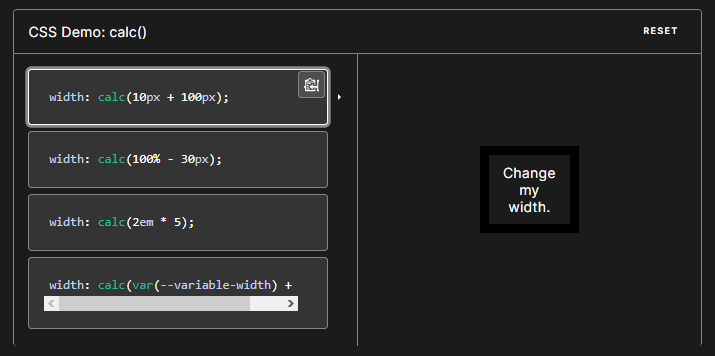
height: calc(100vh - 80px);

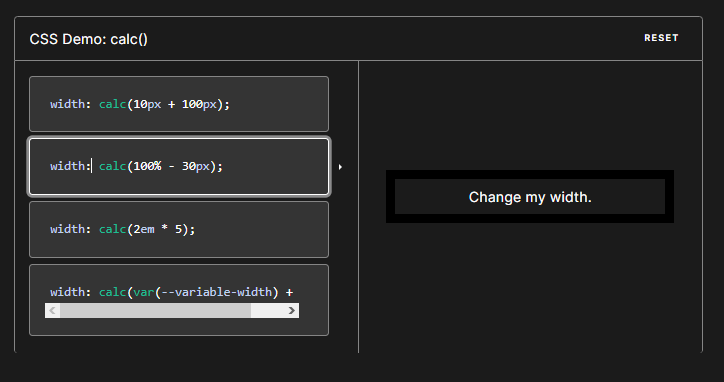
}

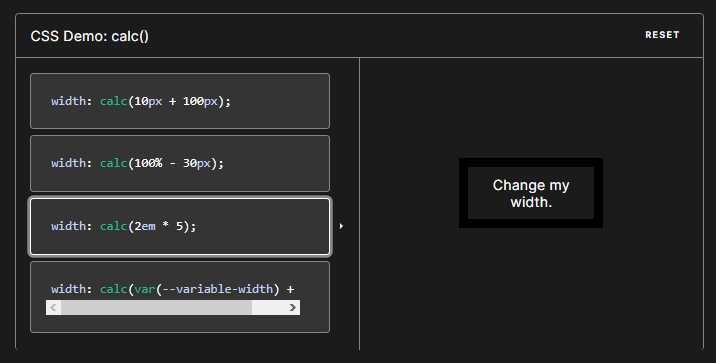
Syntax

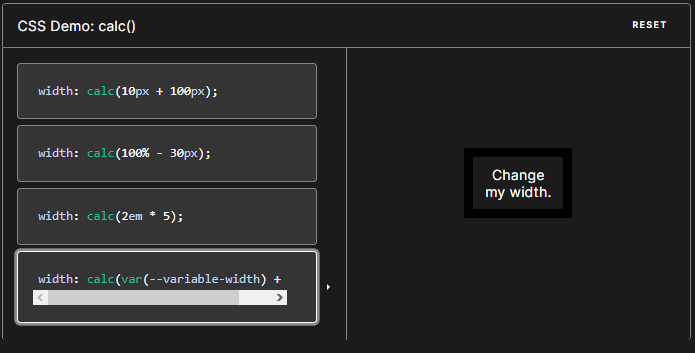
/\* property: calc(expression) \*/

width: calc(100% - 80px)









width: calc(var(--variable-width) + 20px);

**IMP NOTE**

The calc() function takes a single expression as its parameter, with the expression's result used as the value. The expression can be any simple expression combining the following operators, using standard [operator precedence rules](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/Math#operator_precedence):

+

Addition.

-

Subtraction.

\*

Multiplication. At least one of the arguments must be a [<number>](https://developer.mozilla.org/en-US/docs/Web/CSS/number).

/

Division. The right-hand side must be a [<number>](https://developer.mozilla.org/en-US/docs/Web/CSS/number).

The operands in the expression may be any [<length>](https://developer.mozilla.org/en-US/docs/Web/CSS/length) syntax value. You can use different units for each value in your expression, if you wish. You may also use parentheses to establish computation order when needed.

[**Notes**](https://developer.mozilla.org/en-US/docs/Web/CSS/calc#notes)

Serializing the arguments inside calc() follows the IEEE-754 standard for floating point math which means there's a few cases to be aware of regarding the infinity and NaN constants. For more details on how constants are serialized, see the [calc-constant](https://developer.mozilla.org/en-US/docs/Web/CSS/calc-constant) page.

In addition, the following notes apply:

* The + and - operators **must be surrounded by**[**whitespace**](https://developer.mozilla.org/en-US/docs/Glossary/Whitespace). For instance, calc(50% -8px) will be parsed as "a percentage followed by a negative length" — which is an invalid expression — while calc(50% - 8px) is "a percentage followed by a subtraction operator and a length". Likewise, calc(8px + -50%) is treated as "a length followed by an addition operator and a negative percentage".
* The \* and / operators do not require whitespace, but adding it for consistency is recommended.
* Math expressions involving percentages for widths and heights on table columns, table column groups, table rows, table row groups, and table cells in both auto and fixed layout tables *may* be treated as if auto had been specified.
* It is permitted to nest calc() functions, in which case the inner ones are treated as simple parentheses.
* For lengths, you can't use 0 to mean 0px (or another length unit); instead, you must use the version with the unit: margin-top: calc(0px + 20px); is valid, while margin-top: calc(0 + 20px); is invalid.
* The calc() function cannot directly substitute the numeric value for percentage types; for instance calc(100 / 4)% is invalid, while calc(100% / 4) is valid.

**calc() is for values**

The only place you can use the calc() function is in values. See these examples where we’re setting the value for a number of different properties.

.el {

font-size: calc(3vw + 2px);

width: calc(100% - 20px);

height: calc(100vh - 20px);

padding: calc(1vw + 5px);

}

It could be used for only part of a property too, for example:

.el {

margin: 10px calc(2vw + 5px);

border-radius: 15px calc(15px / 3) 4px 2px;

transition: transform calc(1s - 120ms);

}

It can even be a part of another function that forms a part of a property! For example, here’s calc() used within the color stops of a gradient

.el {

background: #1E88E5 linear-gradient(

to bottom,

#1E88E5,

#1E88E5 calc(50% - 10px),

#3949AB calc(50% + 10px),

#3949AB

);

}

There are many [lengths of CSS though](https://css-tricks.com/the-lengths-of-css/), and they can all be used with calc():

* px
* %
* em
* rem
* in
* mm
* cm
* pt
* pc
* ex
* ch
* vh
* vw
* vmin
* vmax

Unit-less numbers are acceptable, too. For example line-height: calc(1.2 \* 1.2); as well as angles like transform: rotate(calc(10deg \* 5));.

**min() function**

The min() CSS function lets you set the smallest (most negative) value from a list of comma-separated values. It can take two parameters and a min function can be used inside another min function if the comparison is to be made between multiple values.

The min() function can be used anywhere a <length>, <frequency>, <angle>, <time>, <percentage>, <number>, or <integer> is allowed.

**IMPORTANT**

In this math function it set the minimum size of the font, if the width/height reduce to it minimum size it does not reduce less than it min size

Syntax

min(value1, value2, ...)

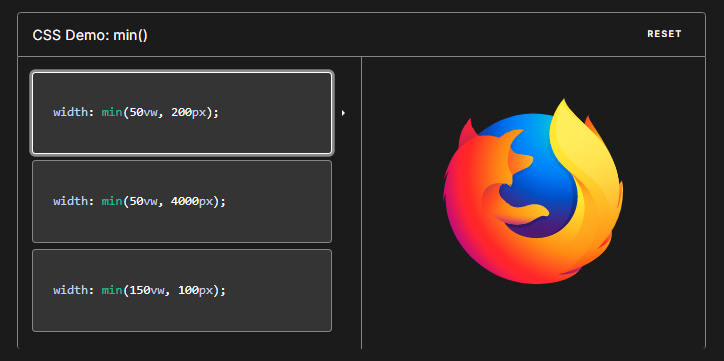
Value

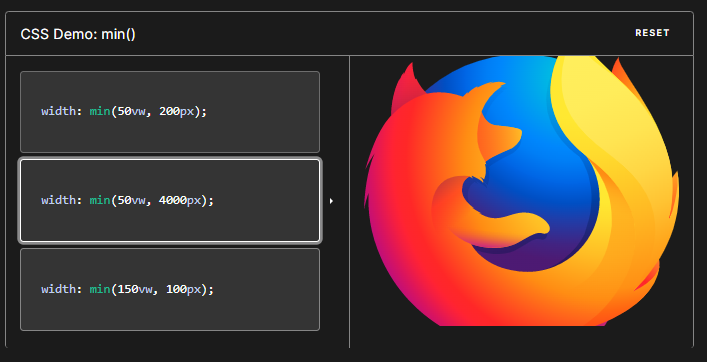
value1, value2, - > Required. A list of comma-separated values - where the smallest value is chosen

min(value1, value2);

min(value1, min(value2, min(value3, value4)));

min(1rem, 50%, 10vw), the browser calculates which of these relative units is the smallest, and uses that value as the actual value.







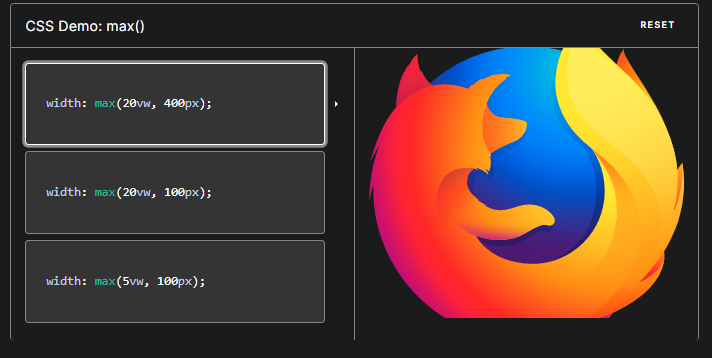
**max() Function**

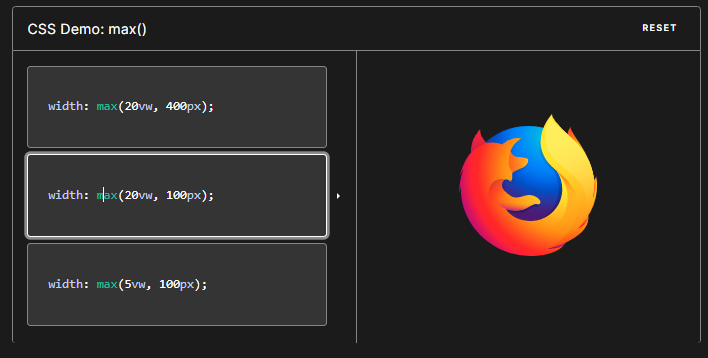
The max() CSS function lets you set the largest (most positive) value from a list of comma-separated expressions as the value of a CSS property value. The max() function can be used anywhere a <length>, <frequency>, <angle>, <time>, <percentage>, <number>, or <integer> is allowed.

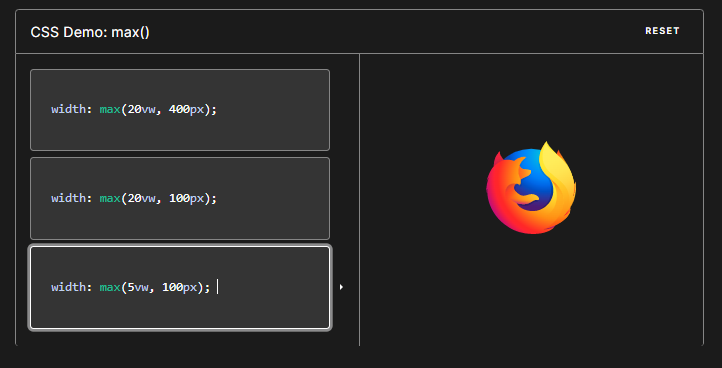
The max() function selects the largest value from a list of comma-separated expressions.

**IMPORTANT**

In this math function it set the maximum size of the font or any Elements, if the width/height increase to it maximum size it does not increase greater than it max size



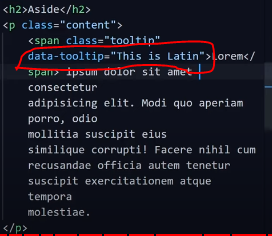




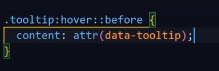
**attr() Function:**

The attr() CSS function is used to retrieve the value of an attribute of the selected element and use it in the stylesheet. It can also be used on pseudo-elements, in which case the value of the attribute on the pseudo-element's originating element is returned.

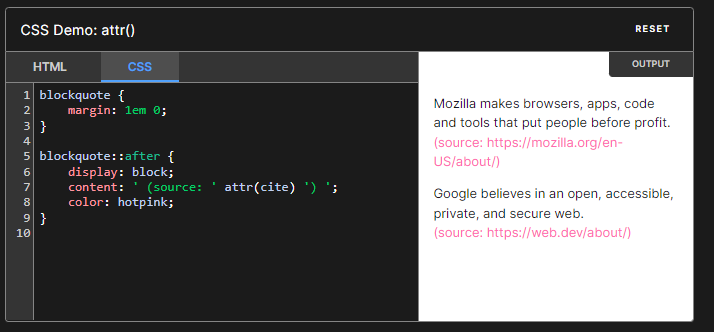
In HMTL we have to put data—name=”NameOfAttrubute”



To access this HTML Attribute we have to use content: attr(data—name);







Syntax

/\* Simple usage \*/

attr(data-count);

attr(title);

/\* With type \*/

attr(src url);

attr(data-count number);

attr(data-width px);

/\* With fallback \*/

attr(data-count number, 0);

attr(src url, "");

attr(data-width px, inherit);

attr(data-something, "default");

**clamp()**

The clamp() CSS function clamps a middle value within a range of values between a defined minimum bound and a maximum bound. The function takes three parameters: a minimum value, a preferred value, and a maximum allowed value.

To use clamp() enter three values: a minimum value, ideal value (from which to calculate), and maximum value.

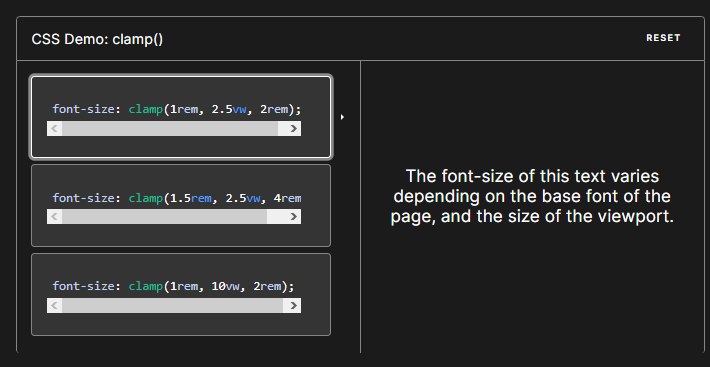
Any of these functions can be used anywhere a <length>, <frequency>, <angle>, <time>, <percentage>, <number>, or <integer> is allowed. You can use these on their own (i.e. font-size: max(0.5vw, 50%, 2rem)), in conjunction with calc() (i.e. font-size: max(calc(0.5vw - 1em), 2rem)), or composed (i.e. font-size: max(min(0.5vw, 1em), 2rem)).

**IMPORTANT**

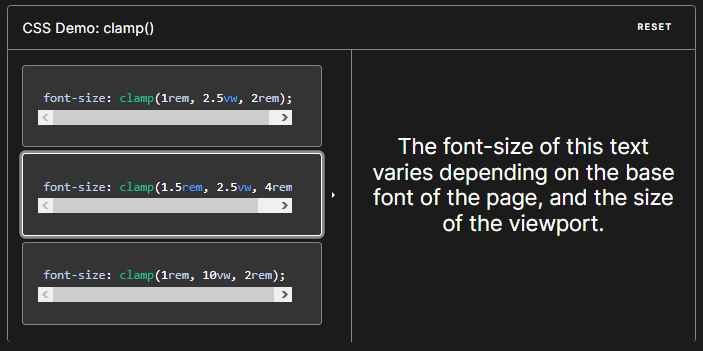
In this math function it set for both minimum & maximum size of the font in left to right in the middle the ideal size of the font, for min() if the width/height reduce to it minimum size it does not reduce less than it min size, for ideal size it maintain the ideal size in between min and max, and for the max, if the width/height increase to it maximum size it does not increase greater than it max size

Syntax

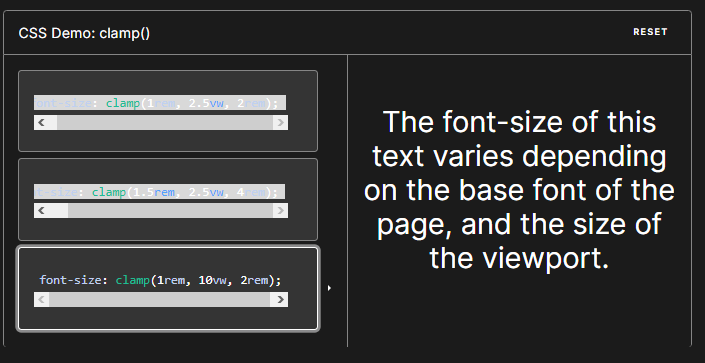
element-name:clamp( minimum size, ideal size, maximum size);



font-size: clamp(1rem, 2.5vw, 2rem);



font-size: clamp(1.5rem, 2.5vw, 4rem);



font-size: clamp(1rem, 10vw, 2rem);

**Syntax**

/\* Static values \*/

width: clamp(200px, 40%, 400px);

width: clamp(20rem, 30vw, 70rem);

width: clamp(10vw, 20em, 100vw);

/\* Calculated values \*/

width: clamp(min(10vw, 20rem), 300px, max(90vw, 55rem));

width: clamp(100px, calc(30% / 2rem + 10px), 900px);

**Parameters**

The clamp(min, ideal, max) function accepts three comma-separated expressions as its parameters.

1. min

The minimum value is the smallest (most negative) value. This is the lower bound in the range of allowed values. If the preferred value is less than this value, the minimum value will be used.

1. val

The preferred value is the expression whose value will be used as long as the result is between the minimum and maximum values.

1. max

The maximum value is the largest (most positive) expression value to which the value of the property will be assigned if the preferred value is greater than this upper bound.

* min(<value-list>): selects the smallest (most negative) value from a list of comma-separated expressions
* max(<value-list>): selects the largest (most positive) value from a list of comma-separated expressions
* clamp(<min>, <ideal>, <max>): clamps a value between an upper and lower bound, based on a set ideal value

**The perfect width**

anything from 45 to 75 characters is widely regarded as a satisfactory length of line for a single-column page set in a serifed text face in a text size."

To ensure that your text blocks are not narrower than 45 characters or wider than 75 characters, use clamp() and the ch (0-width character advance) unit:

p {

width: clamp(45ch, 50%, 75ch);

}

This allows for the browser to determine the width of the paragraph. It will set the width to 50%, unless 50% is smaller than 45ch, at which point 45ch will be selected, and visa versa for if 50% is wider than 75ch. In this demo, the card itself is getting clamped:

**Padding management**

Using the same concept as above, where the min() function can set a "max" value and max() sets a "min" value, you can use max() to set a minimum padding size. This example comes from CSS Tricks, where reader Caluã de Lacerda Pataca shared this idea: The idea is to enable an element to have additional padding at larger screen sizes, but maintain a minimum padding at smaller screen sizes, particularly on the inline padding. To achieve this, use calc() and subtract the minimum padding from either side: calc((100vw - var(--contentWidth)) / 2), or use max: max(2rem, 50vw - var(--contentWidth) / 2). All together it looks like:

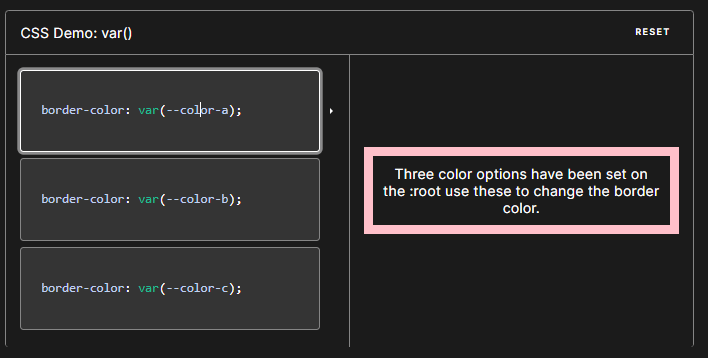
footer {

padding: var(--blockPadding) max(2rem, 50vw - var(--contentWidth) / 2);

}

**var() function**

The var() CSS function can be used to insert the value of a custom property (sometimes called a "CSS variable") instead of any part of a value of another property.

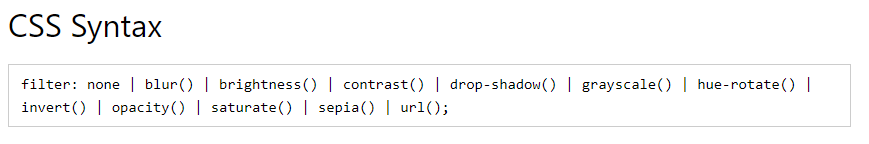


The first argument to the function is the name of the custom property to be substituted. An optional second argument to the function serves as a fallback value. If the custom property referenced by the first argument is invalid, the function uses the second value.

The var() function is used to insert the value of a CSS variable.

CSS variables have access to the DOM, which means that you can create variables with local or global scope, change the variables with JavaScript, and change the variables based on media queries.

## **Filter Functions**



|  |  |  |
| --- | --- | --- |
| **Filter** | **Description** | **Demo** |
| none | Default value. Specifies no effects | [**Demo ❯**](https://www.w3schools.com/cssref/playdemo.php?filename=playcss_filter&preval=none) |
| blur(*px*) | Applies a blur effect to the image. A larger value will create more blur.  If no value is specified, 0 is used. | [**Demo ❯**](https://www.w3schools.com/cssref/playdemo.php?filename=playcss_filter&preval=blur(5px)) |
| brightness(*%*) | Adjusts the brightness of the image.  0% will make the image completely black. 100% (1) is default and represents the original image. Values over 100% will provide brighter results. | [**Demo ❯**](https://www.w3schools.com/cssref/playdemo.php?filename=playcss_filter&preval=brightness(200%25)) |
| contrast(*%*) | Adjusts the contrast of the image.  0% will make the image completely black. 100% (1) is default, and represents the original image. Values over 100% will provide results with more contrast. | [**Demo ❯**](https://www.w3schools.com/cssref/playdemo.php?filename=playcss_filter&preval=contrast(200%25)) |
| drop-shadow(*h-shadow v-shadow blur spread color*) | Applies a drop shadow effect to the image.  Possible values: *h-shadow* - Required. Specifies a pixel value for the horizontal shadow. Negative values place the shadow to the left of the image.  *v-shadow* - Required. Specifies a pixel value for the vertical shadow. Negative values place the shadow above the image.  *blur* - Optional. This is the third value, and must be in pixels. Adds a blur effect to the shadow. A larger value will create more blur (the shadow becomes bigger and lighter). Negative values are not allowed. If no value is specified, 0 is used (the shadow's edge is sharp).  *spread* - Optional. This is the fourth value, and must be in pixels. Positive values will cause the shadow to expand and grow bigger, and negative values will cause the shadow to shrink. If not specified, it will be 0 (the shadow will be the same size as the element). Note: Chrome, Safari and Opera, and maybe other browsers, do not support this 4th length; it will not render if added.  *color* - Optional. Adds a color to the shadow. If not specified, the color depends on the browser (often black).  An example of creating a red shadow, which is 8px big both horizontally and vertically, with a blur effect of 10px:  filter: drop-shadow(8px 8px 10px red);  Tip: This filter is similar to the [box-shadow](https://www.w3schools.com/cssref/css3_pr_box-shadow.php) property. | [**Demo ❯**](https://www.w3schools.com/cssref/playdemo.php?filename=playcss_filter&preval=drop-shadow(8px%208px%2010px%20gray)) |
| grayscale(*%*) | Converts the image to grayscale.  0% (0) is default and represents the original image. 100% will make the image completely gray (used for black and white images).  Note: Negative values are not allowed. | [**Demo ❯**](https://www.w3schools.com/cssref/playdemo.php?filename=playcss_filter&preval=grayscale(100%25)) |
| hue-rotate(*deg*) | Applies a hue rotation on the image. The value defines the number of degrees around the color circle the image samples will be adjusted. 0deg is default, and represents the original image.  Note: Maximum value is 360deg. | [**Demo ❯**](https://www.w3schools.com/cssref/playdemo.php?filename=playcss_filter&preval=hue-rotate(90deg)) |
| invert(*%*) | Inverts the samples in the image.  0% (0) is default and represents the original image. 100% will make the image completely inverted.  Note: Negative values are not allowed. | [**Demo ❯**](https://www.w3schools.com/cssref/playdemo.php?filename=playcss_filter&preval=invert(100%25)) |
| opacity(*%*) | Sets the opacity level for the image. The opacity-level describes the transparency-level, where:  0% is completely transparent. 100% (1) is default and represents the original image (no transparency).  Note: Negative values are not allowed. Tip: This filter is similar to the [opacity](https://www.w3schools.com/cssref/css3_pr_opacity.php) property. | [**Demo ❯**](https://www.w3schools.com/cssref/playdemo.php?filename=playcss_filter&preval=opacity(30%25)) |
| saturate(*%*) | Saturates the image.  0% (0) will make the image completely un-saturated. 100% is default and represents the original image. Values over 100% provides super-saturated results.  Note: Negative values are not allowed. | [**Demo ❯**](https://www.w3schools.com/cssref/playdemo.php?filename=playcss_filter&preval=saturate(8)) |
| sepia(*%*) | Converts the image to sepia.  0% (0) is default and represents the original image. 100% will make the image completely sepia.  Note: Negative values are not allowed. | [**Demo ❯**](https://www.w3schools.com/cssref/playdemo.php?filename=playcss_filter&preval=sepia(100%25)) |
| url() | The url() function takes the location of an XML file that specifies an SVG filter, and may include an anchor to a specific filter element. Example:  filter: url(svg-url#element-id) |  |
| initial | Sets this property to its default value. [Read about *initial*](https://www.w3schools.com/cssref/css_initial.php) |  |
| inherit | Inherits this property from its parent element. [Read about *inherit*](https://www.w3schools.com/cssref/css_inherit.php) |  |

**brightness()**

The brightness() CSS <filter-function> applies a linear multiplier value on an element or an input image, making the image appear brighter or darker.

**Syntax**

brightness(amount)

Values

amount

Brightness specified as a <number> or a <percentage>. A value less than 100% darkens the input image or element, while a value over 100% brightens it. A value of 0% creates a completely black image or element, while a value of 100% leaves the input unchanged. Other values between 0% to 100% have a linear multiplier effect. Values greater than 100% are allowed, providing brighter results. The initial value for interpolation is 1. Negative values are not allowed. The default value, when nothing is specified, is 1.

brightness(0) /\* Brightness is reduced to zero, so input turns black \*/

brightness(0%)

brightness(0.4) /\* Brightness of input is reduced to 40%, so input is 60% darker \*/

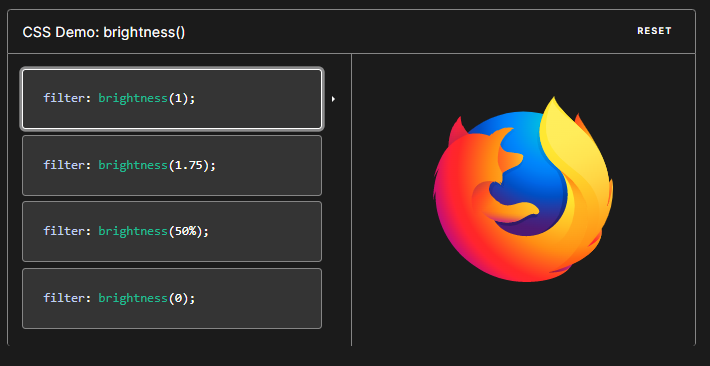
brightness(40%)

brightness(1) /\* Brightness of input is not changed \*/

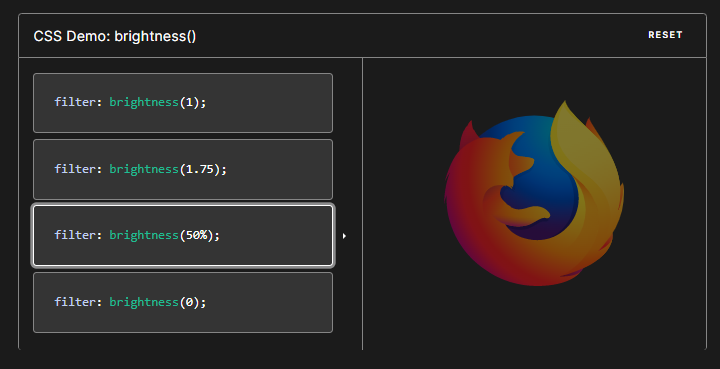
brightness(100%)

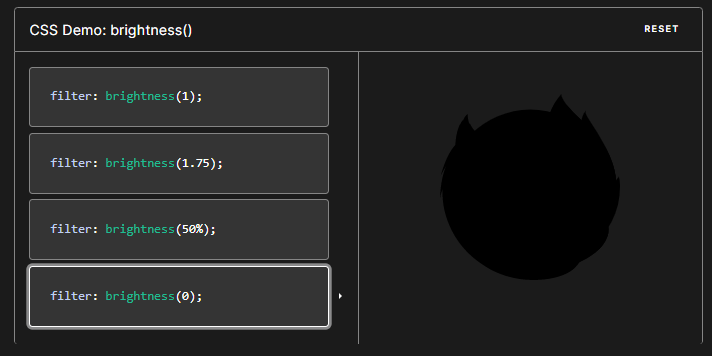
brightness(2) /\* Brightness of input is doubled \*/

brightness(200%)









**hue-rotate()**

The hue-rotate() CSS function rotates the hue of an element and its contents. Its result is a <filter-function>.

**Syntax**

The hue-rotate() function applies a color rotation to the elements on which it is applied.

hue-rotate(angle)

**Values**

angle

The relative change in hue of the input sample, specified as an <angle>. A value of 0deg leaves the input unchanged. A positive hue rotation increases the hue value, while a negative rotation decreases the hue value. The initial value for interpolation is 0. There is no minimum or maximum value. The effect of values above 360deg are, given hue-rotate(Ndeg), evaluates to N modulo 360.

The <angle> CSS data type represents an angle value expressed in degrees, gradians, radians, or turns. The following are equivalent:

CSS

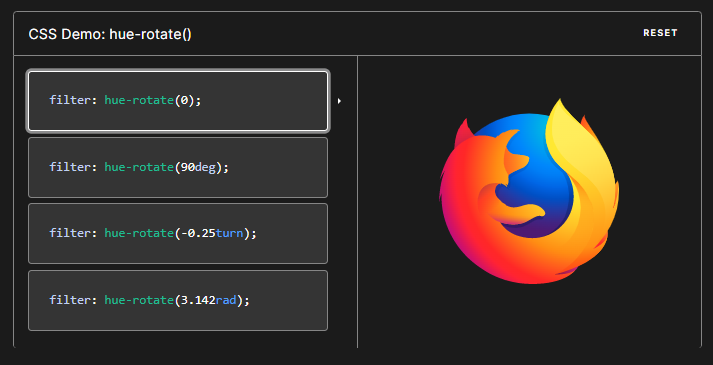
hue-rotate(-180deg)

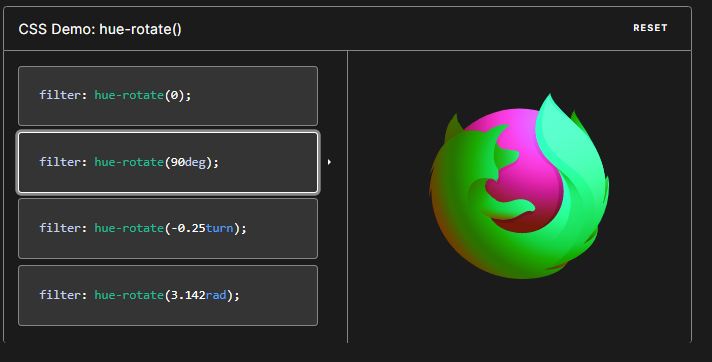
hue-rotate(540deg)

hue-rotate(200grad)

hue-rotate(3.14159rad)

hue-rotate(0.5turn)

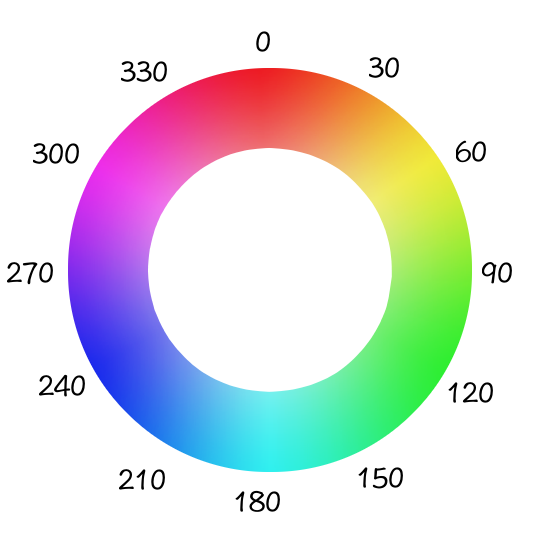








Example for hue:



**repeat()**

The repeat() CSS function represents a repeated fragment of the track list, allowing a large number of columns or rows that exhibit a recurring pattern to be written in a more compact form.

**Syntax**

CSS

/\* <track-repeat> values \*/

repeat(4, 1fr)

repeat(4, [col-start] 250px [col-end])

repeat(4, [col-start] 60% [col-end])

repeat(4, [col-start] 1fr [col-end])

repeat(4, [col-start] min-content [col-end])

repeat(4, [col-start] max-content [col-end])

repeat(4, [col-start] auto [col-end])

repeat(4, [col-start] minmax(100px, 1fr) [col-end])

repeat(4, [col-start] fit-content(200px) [col-end])

repeat(4, 10px [col-start] 30% [col-middle] auto [col-end])

repeat(4, [col-start] min-content [col-middle] max-content [col-end])

/\* <auto-repeat> values \*/

repeat(auto-fill, 250px)

repeat(auto-fit, 250px)

repeat(auto-fill, [col-start] 250px [col-end])

repeat(auto-fit, [col-start] 250px [col-end])

repeat(auto-fill, [col-start] minmax(100px, 1fr) [col-end])

repeat(auto-fill, 10px [col-start] 30% [col-middle] 400px [col-end])

/\* <fixed-repeat> values \*/

repeat(4, 250px)

repeat(4, [col-start] 250px [col-end])

repeat(4, [col-start] 60% [col-end])

repeat(4, [col-start] minmax(100px, 1fr) [col-end])

repeat(4, [col-start] fit-content(200px) [col-end])

repeat(4, 10px [col-start] 30% [col-middle] 400px [col-end])

**The repeat() function takes two arguments:**

repeat count: the first argument specifies the number of times that the track list should be repeated. It is specified with an integer value of 1 or more, or with the keyword values auto-fill or auto-fit. These keyword values repeat the set of tracks as many times as is needed to fill the grid container.

tracks: the second argument specifies the set of tracks that will be repeated. Fundamentally this consists of one or more values, where each value represents the size of that track. Each size is specified using either a <track-size> value or a <fixed-size> value. You can also specify one or more line names before or after each track, by providing <line-names> values before and/or after the track size.

If you use auto-fill or auto-fit to set the repeat count, you may only specify track sizes using the <fixed-size> type, not the <track-size> type. This give us three main syntax forms for repeat():

<track-repeat>, which uses:

an integer to set the repeat count

<track-size> values to set track sizes.

<auto-repeat>, which uses

auto-fill or auto-fit to set the repeat count

<fixed-size> to set track sizes.

<fixed-repeat>, which uses:

an integer to set the repeat count

<fixed-size> values to set track sizes.

Then if a property declaration uses <auto-repeat>, it is only allowed to use <fixed-repeat> for any additional repeat() calls. For example, this is invalid, because it combines the <auto-repeat> form with the <track-repeat> form:

Values

<fixed-size>

One of the following forms:

a <length-percentage> value

a minmax() function with:

min given as a <length-percentage> value

max given as one of a <length-percentage> value, a <flex> value, or one of the following keywords: min-content, max-content, or auto

a minmax() function with:

min given as a <length-percentage> value or one of the following keywords: min-content, max-content, or auto

max given as a <length-percentage> value.

<flex>

A non-negative dimension with the unit fr specifying the track's flex factor. Each <flex>-sized track takes a share of the remaining space in proportion to its flex factor.

<length>

A positive integer length.

<line-names>

Zero or more <custom-ident> values, space-separated and enclosed in square brackets, like this: [first header-start].

<percentage>

A non-negative percentage relative to the inline size of the grid container in column grid tracks, and the block size of the grid container in row grid tracks. If the size of the grid container depends on the size of its tracks, then the <percentage> must be treated as auto. The user-agent may adjust the intrinsic size contributions of the track to the size of the grid container and increase the final size of the track by the minimum amount that would result in honoring the percentage.

<track-size>

One of the following forms:

a <length-percentage> value, a <flex> value, or one of the following keywords: min-content, max-content, or auto

a minmax() function with:

min given as a <length-percentage> value, or one of the following keywords: min-content, max-content, or auto

max given as a <length-percentage> value, a <flex> value, or one of the following keywords: min-content, max-content, or auto

a fit-content() function, passed a <length-percentage> value.

auto

As a maximum, identical to max-content. As a minimum it represents the largest minimum size (as specified by min-width/min-height) of the grid items occupying the grid track.

auto-fill

If the grid container has a definite or maximal size in the relevant axis, then the number of repetitions is the largest possible positive integer that does not cause the grid to overflow its grid container. Treating each track as its maximal track sizing function (each independent value used to define grid-template-rows or grid-template-columns), if that is definite. Otherwise, as its minimum track sizing function, and taking grid-gap into account. If any number of repetitions would overflow, then the repetition is 1. Otherwise, if the grid container has a definite minimal size in the relevant axis, the number of repetitions is the smallest possible positive integer that fulfills that minimum requirement. Otherwise, the specified track list repeats only once.

auto-fit

Behaves the same as auto-fill, except that after placing the grid items any empty repeated tracks are collapsed. An empty track is one with no in-flow grid items placed into or spanning across it. (This can result in all tracks being collapsed, if they're all empty.)

A collapsed track is treated as having a single fixed track sizing function of 0px, and the gutters on either side of it collapse.

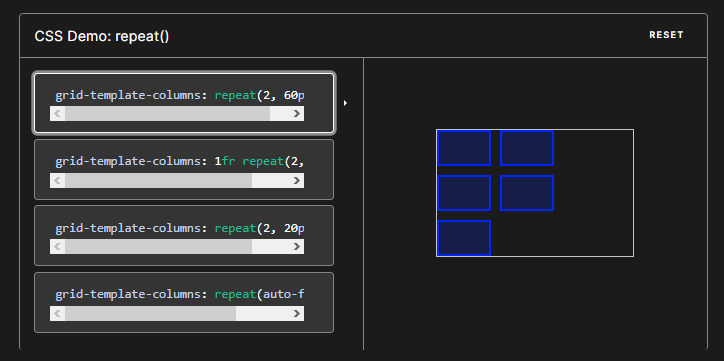
For the purpose of finding the number of auto-repeated tracks, the user agent floors the track size to a user agent specified value (e.g., 1px), to avoid division by zero.

max-content

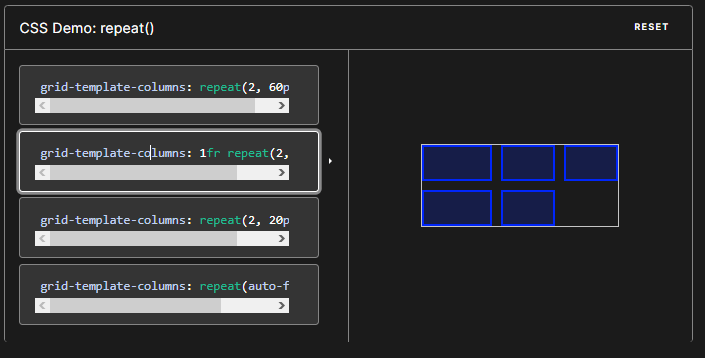
Represents the largest max-content contribution of the grid items occupying the grid track.

min-content

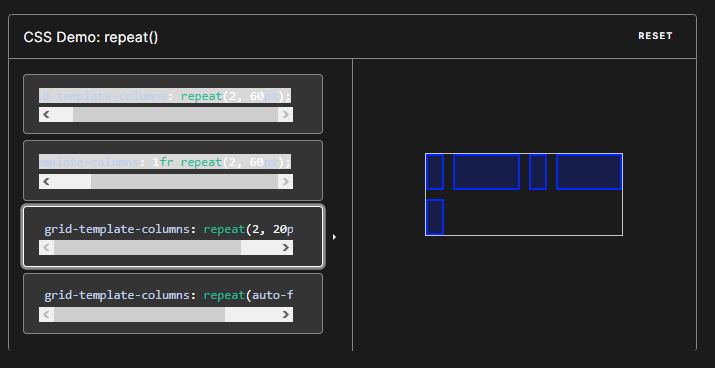
Represents the largest min-content contribution of the grid items occupying the grid track.



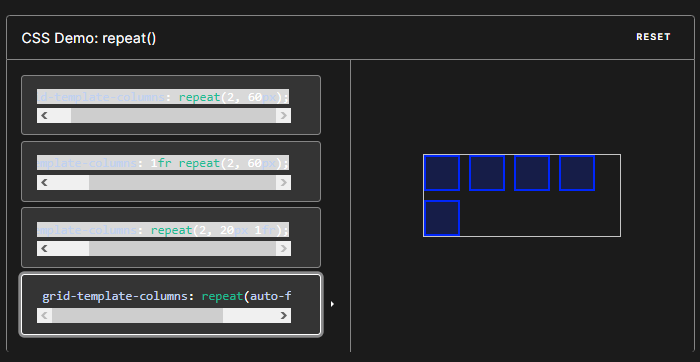
grid-template-columns: repeat(2, 60px);



grid-template-columns: 1fr repeat(2, 60px);



grid-template-columns: repeat(2, 20px 1fr);



grid-template-columns: repeat(auto-fill, 40px);