**CSS Function:**

**Attr():**

Function returns the value of an attribute of an element

Ex:

<style>

1. abbr[title]:after {

content: " (" **attr(title)** ")";

2. a[href]::after {

content: " (" **attr(href)** ")";

3. li:after {

content: " (" **attr(data-sitcom)** ")";

}

</style>

<abbr **title="Big Money Waster**">

BMW

</abbr>

<a **href="https://www.quackit.com**">

Quackit Homepage

</a>

<li **data-sitcom="The Simpsons**">

Bart</li>

O/p:BMW(Big Money Waster)

**Filter properties method:**

**Blur():**

used to apply a blurred effect to an image

Ex:

filter: **blur(3px)**;

**brightness():**

adjust the brightness of an image

Ex:

filter: **brightness(130%);**

filter: **brightness(1.8);**

**Contrast():**

adjust the contrast of an image

Ex:

filter: contrast(140%);

filter: contrast(1.8);

**drop-shadow():**

apply a drop shadow effect to an image

Ex:

filter: drop-shadow(5px 5px 10px gray);

filter: drop-shadow(0px 0px 10px orange);

filter: drop-shadow(5px 5px 0 orange);

**gray-scale();**

convert an image to grayscale

Ex:

filter: grayscale(100%);

filter: grayscale(50%);

filter: grayscale(0.8);

**hue-rotate():**

apply a hue rotation on an image

Ex:

filter: hue-rotate(180deg);

**invert():**

invert the samples in an image

Ex:

filter: invert(70%);

filter: invert(0.8);

**opacity():**

to make an image partially, or completely, transparent

Ex:

filter: opacity(70%);

filter: opacity(0.5);

**saturate():**

adjust the saturation of an image

Ex:

filter: saturate(200%);

filter: saturate(2.5);

**sepia():**

apply a sepia filter to an image

Ex:

filter: sepia(100%);

filter: sepia(50%);

filter: sepia(0.5);

**Height and width Properties Function:**

**calc();**

used in place of other unit types when setting widths, heights, angles, frequencies, etc

Ex:

Width:calc(100% -100px);

width: calc(80% - 1em);

Calc (100% -100px) This calculates into 100 percent of the width, minus of 100px.

**Note:**

addition (+) and subtraction (-) operators must have white space on both sides. There's no such restriction for the multiplication (\*), and division (/) operators.

**Shape-outside and clip-path properties Function:**

**Circle():**

can be used as a value for properties such as shape-outside to control the flow of content around the element

Ex:

shape-outside: circle();

shape-outside: circle(30%);

shape-outside: circle(100px at 10px 150px);

The center of this circle is 10 pixels from the left and 150 pixels from the top

**Ellipse():**

CSS basic shape value that's part of the CSS Shapes module

Ex:

shape-outside: ellipse();

shape-outside: ellipse(30% 50%);

shape-outside: ellipse(100px 50px at 10px 150px);

center of this ellipse is 10 pixels from the left and 150 pixels from the top

clip-path: ellipse();

**inset():**

CSS basic shape value that's part of the CSS Shapes module.

Ex:

/\* Set all four sides \*/

shape-outside: inset(10px);

/\* 10px for top and bottom, 20px for left and right \*/

shape-outside: inset(10px 20px);

/\* 10px for top, 20px for right, 30px for bottom, 15px for left \*/

shape-outside: inset(10px 20px 30px 15px);

**polygon():**

CSS basic shape value that's part of the CSS Shapes module

Ex:

shape-outside: polygon(0px 0px, 200px 100px, 0px 200px);

shape-outside: polygon(0px 0px, 200px 100px, 0px 200px);

**Transition Properties Function:**

**Cubic-bezier():**

used in CSS transitions to create a custom cubic Bézier curve

**Ex:**

transition: width 2s **cubic-bezier(.63,.05,.43,1.7);**

**ease**

Equivalent to cubic-bezier(0.25, 0.1, 0.25, 1).

**linear**

This function is equivalent to cubic-bezier(0, 0, 1, 1).

**ease-in**

This function is equivalent to cubic-bezier(0.42, 0, 1, 1).

**ease-out**

This function is equivalent to cubic-bezier(0, 0, 0.58, 1).

**ease-in-out**

This function is equivalent to cubic-bezier(0.42, 0, 0.58, 1)

**Transform properties function:**

**Matrix():**

used with CSS transforms to style elements in a two-dimensional space

matrix(a, b, c, d, e, f)

(a,d) - scale

(b,c) - skew

(e,f) - translate

Ex:

transform: matrix(0.707107, 0.707107, -0.707107, 0.707107, 150, 0);

(same)

transform: translate(150px, 0px) rotate(45deg);

transform: matrix(2, 0, 0, 2, 0, 0);

(same)

transform: scale(2);

**matrix3d():**

used with CSS transforms to style elements in a three-dimensional space

[rotate3d()](https://www.quackit.com/css/functions/css_rotate3d_function.cfm), [rotateX()](https://www.quackit.com/css/functions/css_rotatex_function.cfm), [rotateY()](https://www.quackit.com/css/functions/css_rotatey_function.cfm), [rotateZ()](https://www.quackit.com/css/functions/css_rotatez_function.cfm), [translate3d()](https://www.quackit.com/css/functions/css_translate3d_function.cfm), [translateZ()](https://www.quackit.com/css/functions/css_translatez_function.cfm), [scale3d()](https://www.quackit.com/css/functions/css_scale3d_function.cfm), [scaleZ()](https://www.quackit.com/css/functions/css_scalez_function.cfm), and [perspective()](https://www.quackit.com/css/functions/css_perspective_function.cfm).

Ex:

transform: rotate3d(1,-2,1,60deg);

**rotate():**

used to rotate elements in a two-dimensional space

Ex:

transform: rotate(15deg);

transform: rotate(20grad);

transform: rotate(.8rad);

transform: rotate(.4turn);

**rotate3d():**

used to rotate elements in a three-dimensional space

Ex:

transform: rotate3d(0, 1, 0, 60deg);

transform: rotate3d(0, 0, 1, 60deg);

transform: rotate3d(1, 5, 1, 60deg);

**rotateY():**

used to rotate elements around the y-axis in a three-dimensional space

Ex:

transform: rotateY(60deg);

transform: rotateY(-60deg);

**rotateZ():**

used to rotate elements around the z-axis in a three-dimensional space

Ex:

transform: rotateZ(60deg);

transform: rotateZ(-60deg);

**scale():**

to scale elements in a two-dimensional space

Ex:

transform: scale(2);

transform: scale(2, 1);

**scaleX():**

to scale elements in a two-dimensional space along the x-axis

Ex:

transform: scaleX(2);

**scaleY():**

to scale elements in a two-dimensional space along the y-axis

Ex:

transform: scaleY(2);

**scaleZ():**

to scale elements in a two-dimensional space along the z-axis

Ex:

transform: scaleZ(2);

**skew():**

to skew elements in a two-dimensional space

Ex:

transform: skew(10deg, 0);

ransform: skew(0, 10deg);

transform: skew(10deg, 10deg);

**skewX():**

to skew elements in a two-dimensional space along the x-axis

Ex:

transform: skewX(10deg);

**skewY():**

to skew elements in a two-dimensional space along the y-axis

Ex:

transform: skewY(10deg)

**translate():**

to move elements in a two-dimensional space

Ex:

transform: translate(60px, 0);

transform: translate(0,60px);

transform: translate(50px, 50px);

**translate3d():**

to move elements in a three-dimensional space

Ex:

transform: translate3d(60px,0,0);

transform: translate3d(0,60px,0);

transform: translate3d(0,0,60px);

**translateX():**

to move elements in a two-dimensional space along the x-axis

Ex:

transform: translateX(60px);

**translateY():**

to move elements in a two-dimensional space along the y-axis

Ex:

transform: translateY(60px);

**translateZ():**

to move elements in a two-dimensional space along the z-axis

Ex:

transform: translateZ(60px);

**url():**

body {

1.background-image: url(/images/image.png);

}

@font-face {

font-family: "Open Sans";

src: url("/fonts/opensans.woff2") format("woff2");

}

@import url("navigation.css");

**Color and background-color properties Function:**

**Hsl(): [hue saturation light]**

used to provide a color value when using CSS. It allows you to specify a color value by specifying the hue, saturation, and light components of the color

Ex:

color: hsl(30, 100%, 50%);

background-color: hsl(120, 100%, 50%);

**hsla():[hue saturation light alpha layer]**

used to add transparency to a color when using the HSL model. It allows you to specify a color value by specifying the hue, saturation, and light components of the color, as well as an alpha layer

Ex:

hsla(30, 100%, 50%, 0.5);

alpha layer : 0-1

**hwb():**

proposed function in CSS that can be used to provide a color value. It allows you to specify a color value by specifying the hue, whiteness, and blackness components of the color, as well as an alpha value

Ex:

color: hwb(240, 0%, 20%);

background-color: hwb(45, 10%, 10%);

**rgb():**

used to provide a color value when using CSS. It allows you to specify an RGB color value by specifying the red, green, and blue channels directly

Ex:

color: rgb(255,0,0);

background-color: rgb(0%,0%,100%);

**rgba():**

used to provide a color value with alpha transparency when using CSS. It allows you to specify an RGB color value, as well as an alpha value to determine the color's transparency

Ex:

rgba(255,0,0,0.5)

rgba(100%,0%,0%,0.5)

**linear-gradient():**

create a linear gradient using CSS

Ex:

background: linear-gradient(purple, yellow);

background: linear-gradient(to right, purple, yellow);

background: linear-gradient(45deg, purple, yellow);

background: linear-gradient(to bottom right, purple, yellow);

background: linear-gradient(to right, red, yellow);

background: linear-gradient(to right, red, green, yellow);

position:

background: linear-gradient(to right, red 5%, green 30%, yellow 75%);

sharp:

background: linear-gradient(to right, red, red 10%, green 10%, green 40%, yellow 40%, yellow);

**radial-gradient():**

allows you to create a radial gradient using CSS

Ex:

background: radial-gradient(yellow, red);

background: radial-gradient(ellipse, black, lime);

background: radial-gradient(circle, black, lime);

background: radial-gradient(circle, black, lime);

background: radial-gradient(yellow, blue, red);

background: radial-gradient(yellow, blue 10%, red 90%);

background: radial-gradient(circle, red, red 20%, green 20%, green 40%, blue 40%, blue 60%, white 60%, black);

**repeating-radial-gradient():**

allows you to create a radial gradient that repeats over and over again infinitely

Ex:

background: repeating-radial-gradient(yellow 20%, red 40%);

background: repeating-radial-gradient(ellipse, green 20%, lime 40%);

background: repeating-radial-gradient(circle, green 20%, lime 40%);

background: repeating-radial-gradient(circle at top left, green 20%, lime 40%);

background: repeating-radial-gradient(circle at top left, lightgreen 20%, green 55%, lime 60%);

background: repeating-radial-gradient(closest-side at 25px 35px, orange 15%, gold 40%);

**repeating-linear-gradient():**

allows you to create a linear gradient that repeats over and over again infinitely in both directions

Ex:

background: repeating-linear-gradient(gold 15%, orange 30%);

background: repeating-linear-gradient(to top right, gold 15%, orange 30%);

background: repeating-linear-gradient(to top right, orange, gold 15%, orange 30%);

background: repeating-linear-gradient(165deg, gold, gold 60px, orange 60px, orange 120px);

**image Element function:**

**image():**

allows you to specify an image with fallback options and annotations

Ex:

*ltr*

image(ltr "arrow.png").

*rtl*

image(rtl "arrow.png").

**Any element Function:**

**Counter():**

enables you to display the counter that has been generated by an element

**Ex:**

<style>

body {

line-height: 0.8em;

**counter-reset: myCounter;**

background: beige;

}

h1 {

font-size: 1.2em;

}

h1:before {

**content: counter(myCounter, lower-roman) ". ";**

**counter-increment: myCounter;**

}

</style>

**Counters():**

function enables you to display nested counters that have been generated by an element and its parent/s

Ex:

<style>

ul {

list-style: none;

**counter-reset: nestedCounter**;

}

ul li {

**counter-increment: nestedCounter**;

line-height: 1.4;

}

ul li:before {

**content: counters(nestedCounter, ".", lower-roman) " - ";**

font-weight: bold;

}

</style>

<ul>

<li>Fruit

<ul>

<li>Apples

<ul>

<li>Green ones</li>

<li>Red ones</li>

</ul>

</li>

<li>Oranges

<ul>

<li>Small ones</li>

<li>Big ones</li>

</ul>

</li>

</ul>

</li>