

CSS Units Ultimate Cheatsheet

- **px** : Absolute Length
- **rem** : Relative to the **font-size** of the root Element
- **em** : Relative to the **font-size** of the Element
- **%** : Relative to the parent Element
- **vw** : Relative to the viewport's width, $1\text{vw} = 1\% \times \text{viewport's width}$
- **vh** : Relative to the viewport's height, $1\text{vh} = 1\% \times \text{viewport's height}$
- **vmin** : Relative to the viewport's smaller dimension, $1\text{vmin} = \min(1\text{vh}, 1\text{vw})$
- **vmax** : Relative to the viewport's larger dimension, $1\text{vmax} = \max(1\text{vh}, 1\text{vw})$
- **ch** : Relative to the width of the glyph "0" of the element's font
- **in** : Inches $1\text{in} = 2.54\text{cm} = 96\text{px}$
- **pc** : Picas $1\text{pc} = 1\text{in} / 6 = 16\text{px}$
- **pt** : Points $1\text{pt} = 1\text{in} / 72 = 1.333\text{px}$ (approximately)
- **cm** : Centimeters $1\text{cm} = 1\text{in} / 2.54 = 37.8\text{px}$ (approximately)
- **mm** : Millimeters $1\text{mm} = 1\text{cm} / 10 = 3.78\text{px}$ (approximately)



CSS Flex Ultimate Cheatsheet

CONTAINER {PARENT} PROPERTIES

DISPLAY

Enables Flex For All Children



`display: flex`



`display: inline-flex`

JUSTIFY-CONTENT

Attempts to distribute extra space on main axis



`flex-start`



`flex-end`



`center`



`space-between`



`space-around`

ALIGN-CONTENT

Only has an effect with more than one line of content.



`flex-start`



`flex-end`



`center`



`space-between`



`space-around`



`stretch`

FLEX-DIRECTION

Establishes the main axis.



`flex-direction:
row`



`flex-direction:
row-reverse`



`flex-direction:
column`



`flex-direction:
column-reverse`

ALIGN-ITEMS

Determine how items are laid out on the cross-axis.



`flex-start`



`flex-end`



`center`



`baseline`



`stretch`

FLEX-WRAP

Wraps items if they can't be made to fit on one line.



`flex-wrap:
no-wrap`



`flex-wrap:
wrap`



`flex-wrap:
wrap-reverse`



CODING.STELLA

CSS Flex Ultimate Cheatsheet

ITEM {CHILDREN} PROPERTIES

FLEX-GROW

Allows you to determine how each child is allowed to grow as a part of a whole.



`flex-grow: 1;`
(Applied to all items)

FLEX-BASIS

Define the size of an element before remaining space is distributed.



`first item 20%;`
`second item 40%`

FLEX-SHRINK

Allows an item to shrink if necessary. Only really useful with a set size or flex-basis.



`both want to be 100% wide, 2nd item has flex-shrink: 2`



`flex-grow: (1, 2 & 3)`

ALIGN-SELF

Sets alignment for individual item.



`3rd item has align-self: flex-end`

ORDER

The order property specifies the order of the flex items.



`order: -1;` on 3rd item



CODING.STELLA

CSS Grid Ultimate Cheatsheet

CONTAINER {PARENT} PROPERTIES

DISPLAY

Establishes a new grid formatting context for children.



`display: grid`



`display: inline-grid`

GRID-TEMPLATE

Defines the rows & columns of the grid.



`grid-template-columns: 14px 14px 14px;
grid-template-rows: 14px 14px 14px;`



`grid-template-columns: repeat(3, 14px);
grid-template-rows: repeat(3, 14px)`



`grid-template-columns: 5px auto 5px;
grid-template-rows: 5px auto 5px;`



`grid-template-columns: 10% 10% auto;
grid-template-rows: 10% 10% auto;`

GRID-GAP

Defines the size of column & row gutters



`grid-gap: 14px;`



`grid-gap: 1px 14px;`



`grid-row-gap: 1px;
grid-column-gap: 14px;`

Note: You can also use `gap` which is very similar to `grid-gap` for both `flexbox` & `grid`.

JUSTIFY-CONTENT

Justifies all grid content on row axis when total grid size is smaller than container.



`justify-content: start`



`justify-content: end`



`justify-content: center`



`justify-content: stretch`



`justify-content: space-around`



`justify-content: space-evenly`



`justify-content: space-between`

ALIGN-CONTENT

Justifies all grid content on column axis when total grid size is smaller than container.



`align-content: start`



`align-content: end`



`align-content: center`



`align-content: stretch`



`align-content: space-around`



`align-content: space-evenly`



`align-content: space-between`

GRID-AUTO-FLOW

Algorithm for automatically placing grid items that aren't explicitly placed.



tells the auto-placement algorithm to fill in each row in turn, adding new rows as necessary

`grid-auto-flow: row`



tells the auto-placement algorithm to fill in each column in turn, adding new columns as necessary

`grid-auto-flow: column`



tells the auto-placement algorithm to attempt to fill in holes earlier in the grid if smaller items come up later

`grid-auto-flow: column`

JUSTIFY-ITEMS

Aligns content in a grid item along the row axis.



`justify-items: start`



`justify-items: end`



`justify-items: center`



`justify-items: stretch
(default)`

ALIGN-ITEMS

Aligns content in a grid item along the column axis.



`align-items: start`



`align-items: end`



`align-items: center`



`align-items: stretch
(default)`

CSS Grid Ultimate Cheatsheet

ITEM {CHILDREN} PROPERTIES

GRID-ROW

Determines an items row-based location within the grid.



```
grid-row-start: 1;  
grid-row-end: 3;
```



```
grid-row-start: span 3
```



```
grid-row-start: 2;  
grid-row-end: 4;
```



```
grid-row: 1 / 3;
```



```
grid-row: 1 / span 3;
```

GRID-COLUMN

Determines an items column-based location within the grid.



```
grid-column-start: 1;  
grid-column-end: 3;
```



```
grid-column-start: span 3;
```



```
grid-column-start: 2;  
grid-column-end: 4;
```



```
grid-column: 2 / 3;
```



```
grid-column: 2 / span 2;
```

GRID-ROW + GRID-COLUMN

Combining grid rows with grid columns.



```
grid-row: 1 / span 1;  
grid-column: 1 / span 1;
```



```
grid-row: 2 / span 2;  
grid-column: 2 / span 2;
```

JUSTIFY-SELF

Aligns content for a specific grid item along the row axis.



```
justify-self: start;
```



```
justify-self: end;
```



```
justify-self: center;
```



```
justify-self: stretch;
```

ALIGN-SELF

Aligns content for a specific grid item along the column axis.



```
align-self: start;
```



```
align-self: end;
```



```
align-self: center;
```



```
align-self: stretch;
```



CODING.STELLA

CSS Filter Ultimate Cheatsheet

```
.selector {  
    filter: blur("2px");  
}
```

PROPERTY VALUE
 ↑
 FUNCTION

Note: You can apply multiple functions but it has to be space separated without comma.



No Filter Applied



filter: blur(2px);



filter: brightness(0.4);



filter: contrast(200%);



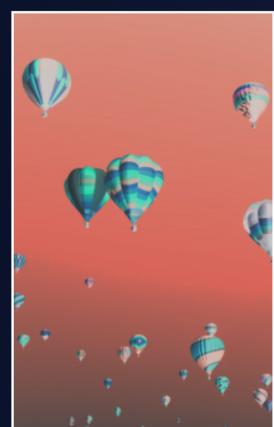
filter: drop-shadow(16px red);



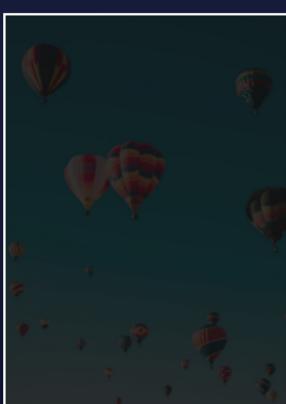
filter: grayscale(80%);



filter: hue-rotate(90deg);



filter: invert(85%);



filter: opacity(15%);



filter: saturate(400%);



filter: sepia(560%);

CSS Animation - Cheatsheet

ANIMATION SHORTHAND PROPERTY

```
animation: dance 300ms linear 100ms infinite alternate-reverse both reverse;
```

animation: **name** **duration** **timing-function** **delay** **count** **direction** **fill-mode** **play-state**

ANIMATION NAME

- Defines which animation keyframe to use.



```
.box {  
background-color: purple;  
border-radius: 10px;  
width: 100px;  
height: 100px;  
  
/* Animation Name */  
animation-name: bounce;  
}
```

If no animation name is specified, no animation is played.

If the name is specified, the keyframes matching the name will be used.

ANIMATION DURATION

- Defines how long the animation lasts.



```
.box {  
background-color: purple;  
border-radius: 10px;  
width: 100px;  
height: 100px;  
  
animation-name: bounce;  
/* Animation Duration */  
animation-duration: 2s;  
}
```

The default value is **zero seconds**: the animation will simply not play.

You can use **decimal values in seconds** with keyword **{s}**

You can use **milliseconds** instead of seconds, with the keyword **{ms}**

ANIMATION TIMING FUNCTION

- Defines how the values between start & the end of the animation are calculated.



```
.box {  
background-color: purple;  
border-radius: 10px;  
width: 100px;  
height: 100px;  
  
animation-name: bounce;  
animation-duration: 2s;  
/* Animation Timing Function */  
animation-timing-function: ease;  
}
```

	Starts	Middle	Ends
ease:	Slow	accelerate	Slow
ease-in:	Slow	→	accelerate gradually
ease-out:	quickly	→	decelerate gradually
ease-in-out:	quickly	→	decelerate gradually
linear:	constant	constant	constant

ANIMATION DELAY

- Defines how long the animation has to wait before starting.



```
.box {  
background-color: purple;  
border-radius: 10px;  
width: 100px;  
height: 100px;  
  
animation-name: bounce;  
animation-duration: 2s;  
animation-timing-function: ease;  
/* Animation Delay */  
animation-delay: 2s;  
}
```

animation-delay: **0s**;

The animation will wait **zero seconds**, and thus start right away.

animation-delay: **1.2s**;

You can use **decimal values in seconds** with the keyword **{s}**.

animation-delay: **2400ms**;

You can use **milliseconds** instead of seconds, with the keyword **{ms}**.



CODING.STELLA

CSS Animation - Cheatsheet

ANIMATION ITERATION COUNT

- * Defines how many times the animation is played.



```
.box {  
background-color: purple;  
border-radius: 10px;  
width: 100px;  
height: 100px;  
  
animation-name: bounce;  
animation-duration: 2s;  
animation-timing-function: ease;  
animation-delay: 2s;  
/* Animation Iteration Count */  
animation-iteration-count: 8;  
}
```

animation-iteration-count: 2s;

You can use **integer values** to define a specific amount of times the animation will play.

animation-iteration-count: infinite;

By using the keyword **infinite**, the animation will play indefinitely.

ANIMATION DIRECTION

- * Defines in which direction the animation is played.



```
.box {  
background-color: purple;  
border-radius: 10px;  
width: 100px;  
height: 100px;  
  
animation-name: bounce;  
animation-duration: 2s;  
animation-timing-function: ease;  
animation-delay: 2s;  
animation-iteration-count: 8;  
/* Animation Direction */  
animation-direction: reverse;  
}
```

animation-direction: normal;

Played **forwards**. When it reaches the end, it starts over at the first keyframe.

animation-direction: reverse;

Played **backwards**: begins at the last keyframe, finishes at the first keyframe.

animation-direction: alternate;

Played **forwards** first, then **backwards**.

animation-direction: alternate-reverse;

Played **backwards** first, then **forwards**.

ANIMATION FILL MODE

- * Defines what happens before an animation starts and after it ends.



```
.box {  
background-color: purple;  
border-radius: 10px;  
width: 100px;  
height: 100px;  
  
animation-name: bounce;  
animation-duration: 2s;  
animation-timing-function: ease;  
animation-delay: 2s;  
animation-iteration-count: 8;  
animation-direction: reverse;  
/* Animation Fill mode */  
animation-fill-mode: backwards;  
}
```

animation-fill-mode: none;

The element is set to its default state before the animation **starts**, and returns to that default state after the animation **ends**.

animation-fill-mode: forwards;

The last styles applied at the end of the animation are retained **afterwards**.

animation-fill-mode: backwards;

The animation's styles will already be applied before the animation actually **starts**.

animation-fill-mode: both;

The styles are applied before and after the **animation plays**.

ANIMATION PLAY STATE

- * Defines if an animation is playing or not.



```
.box {  
background-color: purple;  
border-radius: 10px;  
width: 100px;  
height: 100px;  
  
animation-name: bounce;  
animation-duration: 2s;  
animation-timing-function: ease;  
animation-delay: 2s;  
animation-iteration-count: 8;  
animation-direction: reverse;  
animation-fill-mode: backwards;  
/* Animation Play State */  
animation-play-state: paused;  
}
```

animation-play-state: running;

If the **animation-duration** and **animation-name** are defined, the animation will start playing automatically.

animation-play-state: paused;

The animation is set **paused** at the first keyframe.



CODING.STELLA

CSS Functions - Cheatsheet

CSS functional notation is the type of CSS value that can represent more complex data types or invoke special data processing or calculations.

TRANSFORM FUNCTIONS	MATH FUNCTIONS	FILTER FUNCTIONS	COLOR FUNCTIONS
matrix()	calc()	blur()	hsl()
matrix3d()	clamp()	brightness()	hsla()
perspective()	max()	contrast()	hwb()
rotate()	min()	drop-shadow()	lab()
rotate3d()	abs()	grayscale()	lch()
rotateX()	acos()	hue-rotate()	rgb()
rotateY()	asin()	invert()	rgba()
rotateZ()	atan()	opacity()	color()
scale()	atan2()	saturate()	color-mix()
scaleX()	cos()	sepia()	color-contrast()
scaleY()	exp()		device-cmyk()
scaleZ()	hypot()		
skew()	log()		
skewX()	mod()		
skewY()	pow()		
translate()	rem()		
translate3d()	round()		
translateX()	sign()		
translateY()	sin()		
translateZ()	sqrt()		
	tan()		
REFERENCE FUNCTIONS	IMAGE FUNCTIONS	FONT FUNCTIONS	COUNTER FUNCTIONS
attr()	conic-gradient()	stylistic()	counter()
env()	linear-gradient()	styleset()	counters()
url()	radial-gradient()	character-varient()	symbols()
var()	repeating-linear-gradient()	swash()	
SHAPE FUNCTIONS	repeating-radial-gradient()	ornaments()	
circle()	repeating-conic-gradient()	annotation()	
ellipse()	cross-fade()		
inset()	element()		
polygon()	paint()		
path()			



All CSS Selector Types Explained

ALL CSS SELECTOR TYPES EXPLAINED



* : Universal Selector

- Selects all elements of any type on HTML document.



```
* {  
    color: blue;  
}
```



This rule will change every HTML element on the page to have blue text.



type : type Selector

- Selects elements by node name, selects all elements of the given type within a given document. Ex: `div, span`



```
/* All <a> elements. */  
a {  
    color: red;  
}
```



This rule will change every anchor element on the page to have red color text.



.class-name : Class Selector

- Selects elements based on the content of `class=""` attribute



```
/* All elements with class="spacious" */  
.spacious {  
    margin: 2em;  
}
```



This rule will change add margin of 2em to every element with class attribute of spacious



#id-name : Id Selector

- Selects an element based on the value of its `id=""` attribute.



```
/* The element with id="demo" */  
#demo {  
    padding: 2em;  
}
```



This rule will change add padding of 2em to the element with id attribute of demo



[attr] : Attribute Selector

- matches elements based on the presence or value of a **given attribute**.



```
/* Select <a> element with a title attribute */  
a [title] {  
    color: blue;  
}
```



This rule will change the color of `<a>` element with title attribute to blue.



Grouping Selectors

- Grouping selector `(,)` can select multiple selector at once.



```
/* Selects the div & span and change their color to red */  
div,  
span {  
    color: red;  
}
```



This rule will change the color of `<div>` & `` element to red.



Compound Selectors

- We can combine selectors to increase specificity & readability.



```
/* Only select the <a> element which also has a class of spacious. */  
a.spacious {  
    color: red;  
}
```



This rule will change the color of only a element which also has a class of spacious.



CODING.STELLA

All CSS Combinators Explained

ALL CSS COMBINATORS



Descendant Combinator

- A Descendant Combinator allows us to target a child element.

Select all child & grandchildren of a parent.

```
/* Selects all <strong> elements that  
   are child of p */  
  
p strong {  
  color: red;  
}
```



This rule will change the color of `` elements that are child of `<p>` to blue.



Child Combinator

- Child Combinator helps us to select direct child of the parent.

Selects only direct children & not the grandchildren of a parent.

```
/* Selects all <strong> elements that  
   are direct child of p */  
  
p > strong {  
  color: blue;  
}
```



This rule will change the color of `` elements that are direct child of `<p>` to blue.



General sibling Combinator

- A general sibling works in a way that even if the sibling before the child doesn't the same it will still select it.

```
/* Selects <strong> elements even if there is  
   something different before last <strong> of <p>  
   It will still select it */  
  
p ~ strong {  
  color: red;  
}
```



This rule will change all the `` elements even if the element before the child is not same.



Adjacent Sibling Combinator

- Adjacent sibling works in a way that it will only select a child matches the first element before it.

```
/* Selects <strong> elements only if it  
   matches the first element before it */  
  
p + strong {  
  color: blue;  
}
```



This rule will change the color of `` elements only if matches the first element before it of `<p>`

PSEUDO CLASSES & ELEMENTS SELECTOR:-



: Pseudo Classes

- The `:` pseudo allow the selection of elements based on **state information** that not contained in the **document tree**.

Ex: `a:visited`

```
/* Any button over which the user's  
   pointer is hovering will change  
   its background to blue */  
  
button:hover {  
  color: blue;  
}
```



This rule will change the color of `<button>` element to blue when user pointer is hovered



:: Pseudo Elements

- The `::` pseudo represent entities that are not included in HTML

Ex: `p::first-line`

```
/* The first line of every <p> element */  
p::first-line {  
  color: blue;  
}
```



This rule will change the color of every first line of `p` element to blue



CODING.STELLA



**FOLLOW
US!**

click here!