

 **FREE eBook**

LEARNING CSS

Free unaffiliated eBook created from
Stack Overflow contributors.

#CSS

Table of Contents

About.....	1
Chapter 1: Getting started with CSS.....	2
Remarks.....	2
Versions.....	2
Examples.....	2
External Stylesheet.....	2
Example.....	2
Internal Styles.....	4
Inline Styles.....	4
CSS @import rule (one of CSS at-rule).....	4
How to use @import.....	5
Changing CSS with JavaScript.....	5
Pure JavaScript.....	5
jQuery.....	5
See also.....	6
Styling Lists with CSS.....	6
Chapter 2: 2D Transforms.....	8
Syntax.....	8
Parameters.....	8
Remarks.....	9
2D Coordiante system.....	9
Browser support and prefixes.....	10
Example of prefixed transform:.....	10
Examples.....	10
Rotate.....	10
Scale.....	10
Translate.....	11
Skew.....	11
Multiple transforms.....	12
Transform Origin.....	13

Chapter 3: 3D Transforms	15
Remarks	15
Coordinate system	15
Examples	15
3D cube	16
backface-visibility	17
Compass pointer or needle shape using 3D transforms	18
CSS	18
HTML	18
3D text effect with shadow	19
Chapter 4: Animations	22
Syntax	22
Parameters	22
Examples	22
Animations with the transition property	22
Example	22
Cross-Browser Compatibility	23
Increasing Animation Performance Using the `will-change` Attribute	24
Animations with keyframes	24
Basic Example	24
Cross-browser compatibility	26
Syntax Examples	26
Chapter 5: Backgrounds	27
Introduction	27
Syntax	27
Remarks	27
Examples	27
Background Color	27
Color names	28
Hex color codes	28
RGB / RGBa	28

HSL / HSLa.....	29
Interaction with background-image.....	29
Background Image.....	30
Background Gradients.....	31
linear-gradient()	31
radial-gradient()	31
Repeating gradients	32
Background Shorthand.....	33
Syntax.....	33
Examples.....	33
Background Position.....	34
Longhand Background Position Properties	35
Background Attachment.....	35
Examples	35
background-attachment: scroll.....	35
background-attachment: fixed.....	36
background-attachment: local.....	36
Background Repeat.....	36
Background Color with Opacity.....	37
Multiple Background Image.....	37
The background-origin property.....	38
Background Clip.....	40
Background Size.....	41
General overview	41
Keeping the aspect ratio	43
Eggsplanation for contain and cover.....	43
contain.....	44
cover.....	44
Demonstration with actual code.....	45
background-blend-mode Property.....	46
Chapter 6: Block Formatting Contexts	47

Remarks.....	47
Examples.....	47
Using the overflow property with a value different to visible.....	47
Chapter 7: Border.....	49
Syntax.....	49
Remarks.....	49
Examples.....	51
border-radius.....	51
border-style.....	52
border (shorthands).....	53
border-image.....	53
border-[left right top bottom].....	54
border-collapse.....	54
Multiple Borders.....	54
Creating a multi-colored border using border-image.....	55
CSS.....	55
HTML.....	56
Chapter 8: box-shadow.....	58
Syntax.....	58
Parameters.....	58
Remarks.....	58
Examples.....	58
drop shadow.....	58
inner drop shadow.....	59
bottom-only drop shadow using a pseudo-element.....	59
multiple shadows.....	60
Chapter 9: Browser Support & Prefixes.....	62
Parameters.....	62
Remarks.....	62
Examples.....	62
Transitions.....	62
Transform.....	63

Chapter 10: Cascading and Specificity	64
Remarks	64
Examples	64
Cascading	64
CSS Loading order	64
How are conflicts resolved?	64
Example 1 - Specificity rules	64
Example 2 - Cascade rules with identical selectors	65
Example 3 - Cascade rules after Specificity rules	65
A final note	65
The !important declaration	66
Calculating Selector Specificity	66
Example 1: Specificity of various selector sequences	67
Example 2: How specificity is used by the browser	67
Example 3: How to manipulate specificity	68
!important and inline style declarations	69
A final note	69
More complex specificity example	69
Chapter 11: Centering	71
Examples	71
Using CSS transform	71
CROSS BROWSER COMPATIBILITY	71
MORE INFORMATION	72
Using Flexbox	72
Using position: absolute	73
Ghost element technique (Micha Czernow's hack)	74
Using text-align	74
Centering in relation to another item	75
Vertical align anything with 3 lines of code	76
Vertically align an image inside div	76
Horizontal and Vertical centering using table layout	77
Using calc()	77

Vertically align dynamic height elements.....	78
Using line-height.....	79
Centering vertically and horizontally without worrying about height or width.....	79
The outer container.....	79
The inner container.....	79
The content box.....	79
Demo.....	80
Centering with fixed size.....	80
Horizontal centering with only fixed width.....	81
Vertical centering with fixed height.....	81
Using margin: 0 auto;.....	82
Chapter 12: Clipping and Masking.....	84
Syntax.....	84
Parameters.....	84
Remarks.....	85
Masks:.....	85
Clip-path:.....	85
Examples.....	86
Clipping (Polygon).....	86
CSS:.....	86
HTML:.....	86
Clipping (Circle).....	87
CSS:.....	87
HTML.....	87
Clipping and Masking: Overview and Difference.....	88
Clipping.....	88
Masking.....	88
Simple mask that fades an image from solid to transparent.....	89
CSS.....	89
HTML.....	89
Using masks to cut a hole in the middle of an image.....	90

CSS	90
HTML	90
Using masks to create images with irregular shapes	91
CSS	91
HTML	91
Chapter 13: Colors	93
Syntax	93
Examples	93
Color Keywords	93
Color Keywords	93
Hexadecimal Value	100
Background	100
Syntax	101
rgb() Notation	101
Syntax	102
hsl() Notation	102
Syntax	102
Notes	102
currentColor	103
Use in same element	103
Inherited from parent element	103
rgba() Notation	104
Syntax	104
hsla() Notation	105
Syntax	105
Chapter 14: Columns	106
Syntax	106
Examples	106
Simple Example (column-count)	106
Column Width	107
Chapter 15: Comments	109

Syntax.....	109
Remarks.....	109
Examples.....	109
Single Line.....	109
Multiple Line.....	109
Chapter 16: Counters.....	110
Syntax.....	110
Parameters.....	110
Remarks.....	110
Examples.....	110
Applying roman numerals styling to the counter output.....	111
CSS.....	111
HTML.....	111
Number each item using CSS Counter.....	111
CSS.....	111
HTML.....	112
Implementing multi-level numbering using CSS counters.....	112
CSS.....	112
HTML.....	112
Chapter 17: CSS design patterns.....	114
Introduction.....	114
Remarks.....	114
Examples.....	114
BEM.....	114
Code example.....	115
Chapter 18: CSS Image Sprites.....	116
Syntax.....	116
Remarks.....	116
Examples.....	116
A Basic Implementation.....	116
Chapter 19: CSS Object Model (CSSOM).....	118

Remarks.....	118
Examples.....	118
Introduction.....	118
Adding a background-image rule via the CSSOM.....	118
Chapter 20: Cursor Styling.....	120
Syntax.....	120
Examples.....	120
Changing cursor type.....	120
pointer-events.....	121
caret-color.....	121
Chapter 21: Custom Properties (Variables).....	122
Introduction.....	122
Syntax.....	122
Remarks.....	122
BROWSER SUPPORT / COMPATIBILITY.....	122
Examples.....	122
Variable Color.....	122
Variable Dimensions.....	123
Variable Cascading.....	123
Valid/Invalids.....	124
With media queries.....	125
Chapter 22: Feature Queries.....	127
Syntax.....	127
Parameters.....	127
Remarks.....	127
Examples.....	127
Basic @supports usage.....	127
Chaining feature detections.....	127
Chapter 23: Filter Property.....	129
Syntax.....	129
Parameters.....	129
Remarks.....	130

Examples.....	130
Drop Shadow (use box-shadow instead if possible).....	130
Multiple Filter Values.....	130
Hue Rotate.....	131
Invert Color.....	132
Blur.....	132
Chapter 24: Flexible Box Layout (Flexbox).....	134
Introduction.....	134
Syntax.....	134
Remarks.....	134
Vender Prefixes.....	134
Resources.....	134
Examples.....	135
Sticky Variable-Height Footer.....	135
Holy Grail Layout using Flexbox.....	135
Perfectly aligned buttons inside cards with flexbox.....	137
Dynamic Vertical and Horizontal Centering (align-items, justify-content).....	139
Simple Example (centering a single element).....	139
HTML.....	139
CSS.....	139
Reasoning.....	139
Individual Property Examples.....	140
Example: justify-content: center on a horizontal flexbox.....	140
Example: justify-content: center on a vertical flexbox.....	141
Example: align-content: center on a horizontal flexbox.....	142
Example: align-content: center on a vertical flexbox.....	143
Example: Combination for centering both on horizontal flexbox.....	144
Example: Combination for centering both on vertical flexbox.....	145
Same height on nested containers.....	146
Optimally fit elements to their container.....	147
Chapter 25: Floats.....	149

Syntax.....	149
Remarks.....	149
Examples.....	149
Float an Image Within Text.....	149
Simple Two Fixed-Width Column Layout.....	150
Simple Three Fixed-Width Column Layout.....	151
Two-Column Lazy/Greedy Layout.....	152
clear property.....	153
Clearfix.....	154
Clearfix (with top margin collapsing of contained floats still occurring).....	154
Clearfix also preventing top margin collapsing of contained floats.....	154
Clearfix with support of outdated browsers IE6 and IE7.....	155
In-line DIV using float.....	155
Use of overflow property to clear floats.....	157
Chapter 26: Fragmentation.....	158
Syntax.....	158
Parameters.....	158
Remarks.....	158
Examples.....	158
Media print page-break.....	158
Chapter 27: Functions.....	160
Syntax.....	160
Remarks.....	160
Examples.....	160
calc() function.....	160
attr() function.....	161
linear-gradient() function.....	161
radial-gradient() function.....	161
var() function.....	161
Chapter 28: Grid.....	163
Introduction.....	163

Remarks.....	163
Examples.....	163
Basic Example.....	163
Chapter 29: Inheritance.....	165
Syntax.....	165
Examples.....	165
Automatic inheritance.....	165
Enforced inheritance.....	165
Chapter 30: Inline-Block Layout.....	167
Examples.....	167
Justified navigation bar.....	167
HTML.....	167
CSS.....	167
Notes.....	167
Chapter 31: Internet Explorer Hacks.....	169
Remarks.....	169
Examples.....	169
High Contrast Mode in Internet Explorer 10 and greater.....	169
Examples.....	169
More Information:.....	169
Internet Explorer 6 & Internet Explorer 7 only.....	169
Internet Explorer 8 only.....	170
Adding Inline Block support to IE6 and IE7.....	170
Chapter 32: Layout Control.....	171
Syntax.....	171
Parameters.....	171
Examples.....	172
The display property.....	172
Inline.....	172
Block.....	172
Inline Block.....	172

none	174
To get old table structure using div	174
Chapter 33: Length Units	176
Introduction	176
Syntax	176
Parameters	176
Remarks	177
Examples	177
Font size with rem	177
Creating scalable elements using rems and ems	178
vh and vw	179
vmin and vmax	179
using percent %	179
Chapter 34: List Styles	181
Syntax	181
Parameters	181
Remarks	181
Examples	181
Type of Bullet or Numbering	181
Bullet Position	182
Removing Bullets / Numbers	182
Chapter 35: Margins	184
Syntax	184
Parameters	184
Remarks	184
Examples	184
Apply Margin on a Given Side	184
Direction-Specific Properties	184
Specifying Direction Using Shorthand Property	185
Margin Collapsing	186
Horizontally center elements on a page using margin	188

Margin property simplification.....	188
Negative margins.....	189
Example 1:.....	189
Chapter 36: Media Queries.....	191
Syntax.....	191
Parameters.....	191
Remarks.....	192
Examples.....	193
Basic Example.....	193
Use on link tag.....	193
mediatype.....	193
Using Media Queries to Target Different Screen Sizes.....	194
Width vs Viewport.....	195
Media Queries for Retina and Non Retina Screens.....	196
Terminology and Structure.....	196
General Structure of a Media Query.....	196
A Media Query containing a Media Type.....	196
A Media Query containing a Media Type and a Media Feature.....	197
A Media Query containing a Media Feature (and an implicit Media Type of "all").....	197
Media queries and IE8.....	197
A Javascript based workaround.....	197
The alternative.....	198
Chapter 37: Multiple columns.....	199
Introduction.....	199
Remarks.....	199
Examples.....	199
Basic example.....	199
Create Multiple Columns.....	200
Chapter 38: Normalizing Browser Styles.....	201
Introduction.....	201
Remarks.....	201

Examples.....	201
normalize.css.....	201
What does it do.....	201
Difference to reset.css.....	202
Approaches and Examples.....	202
Chapter 39: Object Fit and Placement.....	204
Remarks.....	204
Examples.....	204
object-fit.....	204
Chapter 40: Opacity.....	207
Syntax.....	207
Remarks.....	207
Examples.....	207
Opacity Property.....	207
IE Compatibility for `opacity`.....	207
Chapter 41: Outlines.....	209
Syntax.....	209
Parameters.....	209
Remarks.....	209
Examples.....	210
Overview.....	210
outline-style.....	210
Chapter 42: Overflow.....	212
Syntax.....	212
Parameters.....	212
Remarks.....	212
Examples.....	212
overflow: scroll.....	212
overflow-wrap.....	213
overflow: visible.....	214
Block Formatting Context Created with Overflow.....	215
overflow-x and overflow-y.....	216

Chapter 43: Padding	218
Syntax	218
Remarks	218
Examples	218
Padding on a given side	218
Padding Shorthand	219
Chapter 44: Performance	221
Examples	221
Use transform and opacity to avoid trigger layout	221
DON'T	221
DO	222
Chapter 45: Positioning	223
Syntax	223
Parameters	223
Remarks	223
Examples	223
Fixed position	223
Overlapping Elements with z-index	224
Example	224
HTML	224
CSS	224
Syntax	225
Remarks	225
Relative Position	226
Absolute Position	226
Static positioning	226
Chapter 46: Pseudo-Elements	228
Introduction	228
Syntax	228
Parameters	228
Remarks	228

Examples.....	229
Pseudo-Elements.....	229
Pseudo-Elements in Lists.....	229
Chapter 47: Selectors.....	231
Introduction.....	231
Syntax.....	231
Remarks.....	231
Examples.....	231
Attribute Selectors.....	231
Overview.....	231
Details.....	232
[attribute].....	232
[attribute="value"].....	233
[attribute*="value"].....	233
[attribute~="value"].....	233
[attribute^="value"].....	234
[attribute\$="value"].....	234
[attribute ="value"].....	234
[attribute="value" i].....	234
Specificity of attribute selectors.....	235
0-1-0.....	235
Combinators.....	235
Overview.....	235
Descendant Combinator: selector selector.....	235
Child Combinator: selector > selector.....	236
Adjacent Sibling Combinator: selector + selector.....	236
General Sibling Combinator: selector ~ selector.....	237
Class Name Selectors.....	237
ID selectors.....	238
Pseudo-classes.....	239
Syntax.....	239

List of pseudo-classes:.....	239
Basic selectors.....	242
How to style a Range input.....	242
Global boolean with checkbox:checked and ~ (general sibling combinator).....	243
Add boolean as a checkbox.....	243
Change the boolean's value.....	243
Accessing boolean value with CSS.....	243
In action.....	244
CSS3 :in-range selector example.....	244
Child Pseudo Class.....	244
Select element using its ID without the high specificity of the ID selector.....	245
A. The :not pseudo-class example & B. :focus-within CSS pseudo-class.....	245
The :only-child pseudo-class selector example.....	247
The :last-of-type selector.....	248
Chapter 48: Shapes for Floats.....	249
Syntax.....	249
Parameters.....	249
Remarks.....	249
Examples.....	249
Shape Outside with Basic Shape – circle().....	249
Shape margin.....	251
Chapter 49: Single Element Shapes.....	252
Examples.....	252
Square.....	252
Triangles.....	252
Bursts.....	256
Circles and Ellipses.....	257
Circle.....	257
Ellipse.....	258
Trapezoid.....	258
Cube.....	259

Pyramid.....	260
Chapter 50: Stacking Context.....	262
Examples.....	262
Stacking Context.....	262
Chapter 51: Structure and Formatting of a CSS Rule.....	266
Remarks.....	266
Good.....	266
Bad.....	266
One-Liner.....	266
Examples.....	266
Rules, Selectors, and Declaration Blocks.....	266
Property Lists.....	266
Multiple Selectors.....	267
Chapter 52: Tables.....	268
Syntax.....	268
Remarks.....	268
Examples.....	268
table-layout.....	268
border-collapse.....	269
border-spacing.....	269
empty-cells.....	270
caption-side.....	270
Chapter 53: The Box Model.....	272
Syntax.....	272
Parameters.....	272
Remarks.....	272
About padding-box.....	272
Examples.....	272
What is the Box Model?.....	272
The Edges.....	272
Example.....	273

box-sizing.....	274
Chapter 54: Transitions.....	277
Syntax.....	277
Parameters.....	277
Remarks.....	277
Examples.....	277
Transition shorthand.....	277
Transition (longhand).....	278
CSS.....	278
HTML.....	278
cubic-bezier.....	279
Chapter 55: Typography.....	281
Syntax.....	281
Parameters.....	281
Remarks.....	282
Examples.....	282
Font Size.....	282
The Font Shorthand.....	282
Font Stacks.....	283
Letter Spacing.....	283
Text Transform.....	284
Text Indent.....	284
Text Decoration.....	285
Text Overflow.....	285
Word Spacing.....	286
Text Direction.....	286
Font Variant.....	287
Quotes.....	288
Text Shadow.....	288
Shadow without blur radius.....	288
Shadow with blur radius.....	288
Multiple Shadows.....	288

Chapter 56: Vertical Centering	289
Remarks.....	289
Examples.....	289
Centering with display: table.....	289
Centering with Transform.....	289
Centering with Flexbox.....	290
Centering Text with Line Height.....	290
Centering with Position: absolute.....	291
Centering with pseudo element.....	292
Credits	293

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: [css](#)

It is an unofficial and free CSS ebook created for educational purposes. All the content is extracted from [Stack Overflow Documentation](#), which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official CSS.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with CSS

Remarks

Styles can be authored in several ways, allowing for varying degrees of reuse and scope when they are specified in a source HTML document. *External* stylesheets can be reused across HTML documents. *Embedded* stylesheets apply to the entire document in which they are specified. *Inline* styles apply only to the individual HTML element on which they are specified.

Versions

Version	Release Date
1	1996-12-17
2	1998-05-12
3	2015-10-13

Examples

External Stylesheet

An external CSS stylesheet can be applied to any number of HTML documents by placing a `<link>` element in each HTML document.

The attribute `rel` of the `<link>` tag has to be set to `"stylesheet"`, and the `href` attribute to the relative or absolute path to the stylesheet. While using relative URL paths is generally considered good practice, absolute paths can be used, too. In HTML5 the `type` attribute [can be omitted](#).

It is recommended that the `<link>` tag be placed in the HTML file's `<head>` tag so that the styles are loaded before the elements they style. Otherwise, [users will see a flash of unstyled content](#).

Example

hello-world.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <link rel="stylesheet" type="text/css" href="style.css">
  </head>
  <body>
    <h1>Hello world!</h1>
```



```
<p>I ♥ CSS</p>
</body>
</html>
```

style.css

```
h1 {
  color: green;
  text-decoration: underline;
}
p {
  font-size: 25px;
  font-family: 'Trebuchet MS', sans-serif;
}
```

Make sure you include the correct path to your CSS file in the href. If the CSS file is in the same folder as your HTML file then no path is required (like the example above) but if it's saved in a folder, then specify it like this `href="foldername/style.css"`.

```
<link rel="stylesheet" type="text/css" href="foldername/style.css">
```

External stylesheets are considered the best way to handle your CSS. There's a very simple reason for this: when you're managing a site of, say, 100 pages, all controlled by a single stylesheet, and you want to change your link colors from blue to green, it's a lot easier to make the change in your CSS file and let the changes "cascade" throughout all 100 pages than it is to go into 100 separate pages and make the same change 100 times. Again, if you want to completely change the look of your website, you only need to update this one file.

You can load as many CSS files in your HTML page as needed.

```
<link rel="stylesheet" type="text/css" href="main.css">
<link rel="stylesheet" type="text/css" href="override.css">
```

CSS rules are applied with some basic rules, and order does matter. For example, if you have a `main.css` file with some code in it:

```
p.green { color: #00FF00; }
```

All your paragraphs with the 'green' class will be written in light green, but you can override this with another .css file just by including it *after* `main.css`. You can have `override.css` with the following code follow `main.css`, for example:

```
p.green { color: #006600; }
```

Now all your paragraphs with the 'green' class will be written in darker green rather than light green.

Other principles apply, such as the 'important' rule, specificity, and inheritance.

When someone first visits your website, their browser downloads the HTML of the current page plus the linked CSS file. Then when they navigate to another page, their browser only needs to download the HTML of that page; the CSS file is cached, so it does not need to be downloaded again. Since browsers cache the external stylesheet, your pages load faster.

Internal Styles

CSS enclosed in `<style></style>` tags within an HTML document functions like an external stylesheet, except that it lives in the HTML document it styles instead of in a separate file, and therefore can only be applied to the document in which it lives. Note that this element *must* be inside the `<head>` element for HTML validation (though it will work in all current browsers if placed in `body`).

```
<head>
  <style>
    h1 {
      color: green;
      text-decoration: underline;
    }
    p {
      font-size: 25px;
      font-family: 'Trebuchet MS', sans-serif;
    }
  </style>
</head>
<body>
  <h1>Hello world!</h1>
  <p>I ♥ CSS</p>
</body>
```

Inline Styles

Use inline styles to apply styling to a specific element. Note that this is **not** optimal. Placing style rules in a `<style>` tag or external CSS file is encouraged in order to maintain a distinction between content and presentation.

Inline styles override any CSS in a `<style>` tag or external style sheet. While this can be useful in some circumstances, this fact more often than not reduces a project's maintainability.

The styles in the following example apply directly to the elements to which they are attached.

```
<h1 style="color: green; text-decoration: underline;">Hello world!</h1>
<p style="font-size: 25px; font-family: 'Trebuchet MS';">I ♥ CSS</p>
```

Inline styles are generally the safest way to ensure rendering compatibility across various email clients, programs and devices, but can be time-consuming to write and a bit challenging to manage.

CSS @import rule (one of CSS at-rule)

The `@import` CSS at-rule is used to import style rules from other style sheets. These rules must

precede all other types of rules, except `@charset` rules; as it is not a nested statement, `@import` cannot be used inside conditional group at-rules. [@import](#).

How to use @import

You can use `@import` rule in following ways:

A. With internal style tag

```
<style>
  @import url('/css/styles.css');
</style>
```

B. With external stylesheet

The following line imports a CSS file named `additional-styles.css` in the root directory into the CSS file in which it appears:

```
@import '/additional-styles.css';
```

Importing external CSS is also possible. A common use case are font files.

```
@import 'https://fonts.googleapis.com/css?family=Lato';
```

An optional second argument to `@import` rule is a list of media queries:

```
@import '/print-styles.css' print;
@import url('landscape.css') screen and (orientation:landscape);
```

Changing CSS with JavaScript

Pure JavaScript

It's possible to add, remove or change CSS property values with JavaScript through an element's `style` property.

```
var el = document.getElementById("element");
el.style.opacity = 0.5;
el.style.fontFamily = 'sans-serif';
```

Note that style properties are named in lower camel case style. In the example you see that the css property `font-family` becomes `fontFamily` in javascript.

As an alternative to working directly on elements, you can create a `<style>` or `<link>` element in JavaScript and append it to the `<body>` or `<head>` of the HTML document.

jQuery

Modifying CSS properties with jQuery is even simpler.

```
$('#element').css('margin', '5px');
```

If you need to change more than one style rule:

```
$('#element').css({  
    margin: "5px",  
    padding: "10px",  
    color: "black"  
});
```

jQuery includes two ways to change css rules that have hyphens in them (i.e. `font-size`). You can put them in quotes or camel-case the style rule name.

```
$('.example-class').css({  
    "background-color": "blue",  
    fontSize: "10px"  
});
```

See also

- [JavaScript documentation – Reading and Changing CSS Style.](#)
- [jQuery documentation – CSS Manipulation](#)

Styling Lists with CSS

There are three different properties for styling list-items: `list-style-type`, `list-style-image`, and `list-style-position`, which should be declared in that order. The default values are `disc`, `outside`, and `none`, respectively. Each property can be declared separately, or using the `list-style` shorthand property.

`list-style-type` defines the shape or type of bullet point used for each list-item.

Some of the acceptable values for `list-style-type`:

- `disc`
- `circle`
- `square`
- `decimal`
- `lower-roman`
- `upper-roman`
- `none`

(For an exhaustive list, see the [W3C specification wiki](#))

To use square bullet points for each list-item, for example, you would use the following property-value pair:

```
li {  
  list-style-type: square;  
}
```

The **list-style-image** property determines whether the list-item icon is set with an image, and accepts a value of `none` or a URL that points to an image.

```
li {  
  list-style-image: url(images/bullet.png);  
}
```

The **list-style-position** property defines where to position the list-item marker, and it accepts one of two values: "inside" or "outside".

```
li {  
  list-style-position: inside;  
}
```

Read Getting started with CSS online: <https://riptutorial.com/css/topic/293/getting-started-with-css>

Chapter 2: 2D Transforms

Syntax

- **Rotate Transform**
 - transform: rotate(<angle>)
- **Translate Transform**
 - transform: translate(<length-or-percentage> [, <length-or-percentage>]?)
 - transform: translateX(<length-or-percentage>)
 - transform: translateY(<length-or-percentage>)
- **Skew Transform**
 - transform: skew(<angle> [, <angle>]?)
 - transform: skewX(<angle>)
 - transform: skewY(<angle>)
- **Scale Transform**
 - transform: scale(<scale-factor> [, <scale-factor>]?)
 - transform: scaleX(<scale-factor>)
 - transform: scaleY(<scale-factor>)
- **Matrix Transform**
 - transform: matrix(<number> [, <number>]{5,5})

Parameters

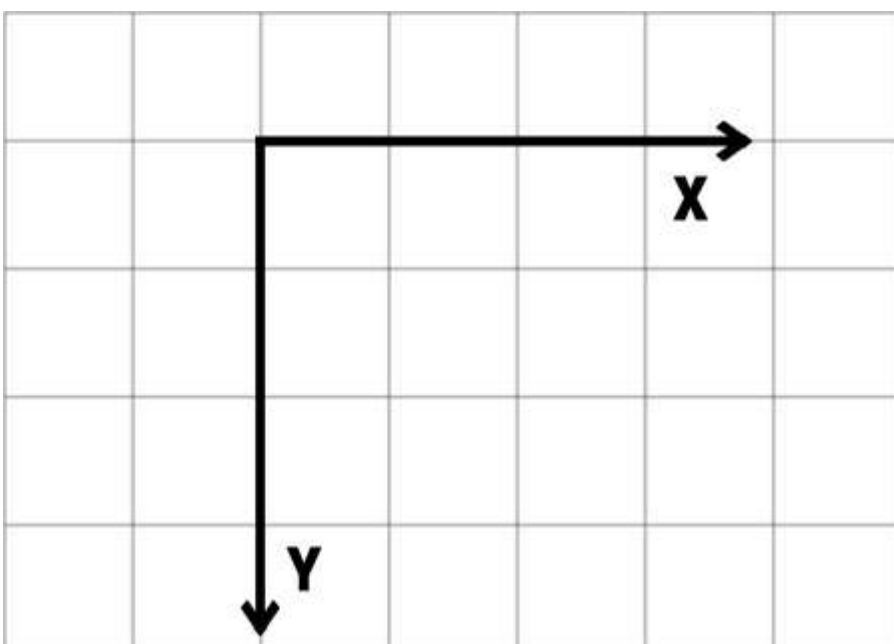
Function/Parameter	Details
rotate(x)	Defines a transformation that moves the element around a fixed point on the Z axis
translate(x, y)	Moves the position of the element on the X and Y axis
translateX(x)	Moves the position of the element on the X axis
translateY(y)	Moves the position of the element on the Y axis
scale(x, y)	Modifies the size of the element on the X and Y axis
scaleX(x)	Modifies the size of the element on the X axis
scaleY(y)	Modifies the size of the element on the Y axis
skew(x, y)	Shear mapping, or transvection, distorting each point of an element by a certain angle in each direction
skewX(x)	Horizontal shear mapping distorting each point of an element by a certain angle in the horizontal direction

Function/Parameter	Details
<code>skewY(y)</code>	Vertical shear mapping distorting each point of an element by a certain angle in the vertical direction
<code>matrix()</code>	Defines a 2D transformation in the form of a transformation matrix.
angle	The angle by which the element should be rotated or skewed (depending on the function with which it is used). Angle can be provided in degrees (<code>deg</code>), gradians (<code>grad</code>), radians (<code>rad</code>) or turns (<code>turn</code>). In <code>skew()</code> function, the second angle is optional. If not provided, there will be no (0) skew in Y-axis.
length-or-percentage	The distance expressed as a length or a percentage by which the element should be translated. In <code>translate()</code> function, the second length-or-percentage is optional. If not provided, then there would be no (0) translation in Y-axis.
scale-factor	A number which defines how many times the element should be scaled in the specified axis. In <code>scale()</code> function, the second scale-factor is optional. If not provided, the first scale-factor will be applied for Y-axis also.

Remarks

2D Coördinate system

Transforms are made according to a 2D X/Y coordinate system. The X axis goes from right to left and the Y axis goes downwards as shown in the following image:



So a positive `translateY()` goes downwards and a positive `translateX()` goes right.

Browser support and prefixes

- IE supports this property since IE9 with the `-ms-` prefix. Older versions and Edge don't need the prefix
- Firefox supports it since version 3.5 and needs the `-moz-` prefix until version 15
- Chrome since version 4 and until version 34 needs the `-webkit-` prefix
- Safari needs the `-webkit-` prefix until version 8
- Opera needs the `-o-` prefix for version 11.5 and the `-webkit-` prefix from version 15 to 22
- Android needs the `-webkit-` prefix from version 2.1 to 4.4.4

Example of prefixed transform:

```
-webkit-transform: rotate(45deg);  
-ms-transform: rotate(45deg);  
transform: rotate(45deg);
```

Examples

Rotate

HTML

```
<div class="rotate"></div>
```

CSS

```
.rotate {  
  width: 100px;  
  height: 100px;  
  background: teal;  
  transform: rotate(45deg);  
}
```

This example will rotate the div by 45 degrees clockwise. The center of rotation is in the center of the div, 50% from left and 50% from top. You can change the center of rotation by setting the `transform-origin` property.

```
transform-origin: 100% 50%;
```

The above example will set the center of rotation to the middle of the right side end.

Scale

HTML

```
<div class="scale"></div>
```


CSS

```
.scale {  
  width: 100px;  
  height: 100px;  
  background: teal;  
  transform: scale(0.5, 1.3);  
}
```

This example will scale the div to $100\text{px} * 0.5 = 50\text{px}$ on the X axis and to $100\text{px} * 1.3 = 130\text{px}$ on the Y axis.

The center of the transform is in the center of the div, 50% from left and 50% from top.

Translate

HTML

```
<div class="translate"></div>
```

CSS

```
.translate {  
  width: 100px;  
  height: 100px;  
  background: teal;  
  transform: translate(200px, 50%);  
}
```

This example will move the div by 200px on the X axis and by $100\text{px} * 50\% = 50\text{px}$ on the Y axis.

You can also specify translations on a single axis.

On the X axis:

```
.translate {  
  transform: translateX(200px);  
}
```

On the Y axis:

```
.translate {  
  transform: translateY(50%);  
}
```

Skew

HTML

```
<div class="skew"></div>
```

CSS

```
.skew {  
  width: 100px;  
  height: 100px;  
  background: teal;  
  transform: skew(20deg, -30deg);  
}
```

This example will skew the div by 20 degrees on the X axis and by - 30 degrees on the Y axis. The center of the transform is in the center of the div, 50% from left and 50% from top.

See the result [here](#).

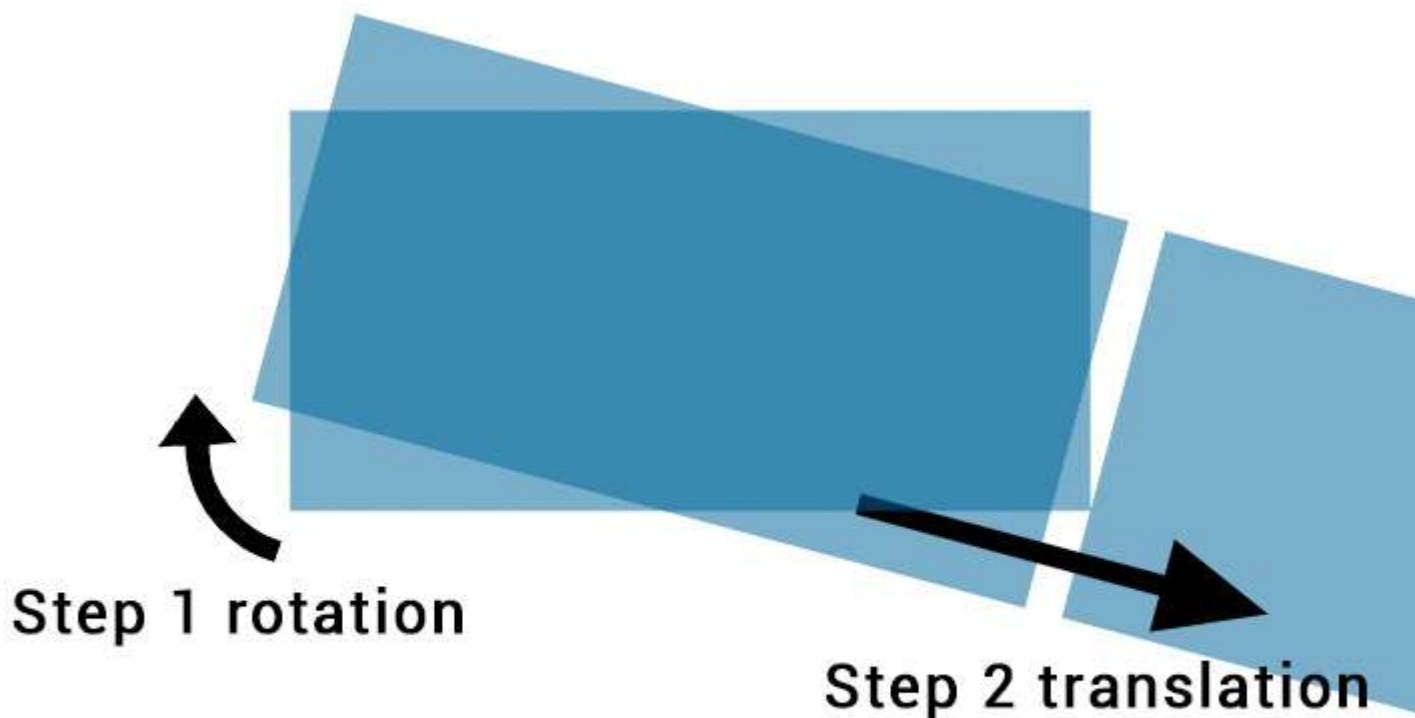
Multiple transforms

Multiple transforms can be applied to an element in one property like this:

```
transform: rotate(15deg) translateX(200px);
```

This will rotate the element 15 degrees clockwise and then translate it 200px to the right.

In chained transforms, **the coordinate system moves with the element**. This means that the translation won't be horizontal but on an axis rotate 15 degrees clockwise as shown in the following image:



Changing the order of the transforms will change the output. The first example will be different to

```
transform: translateX(200px) rotate(15deg);
```

```
<div class="transform"></div>
```

```
.transform {  
  transform: rotate(15deg) translateX(200px);  
}
```

As shown in this image:



Step 1 translation

Transform Origin

Transformations are done with respect to a point which is defined by the `transform-origin` property.

The property takes 2 values : `transform-origin: X Y;`

In the following example the first div (`.tl`) is rotate around the top left corner with `transform-origin: 0 0;` and the second (`.tr`) is transformed around it's top right corner with `transform-origin: 100% 0.`

The rotation is applied **on hover** :

HTML:

```
<div class="transform origin1"></div>
<div class="transform origin2"></div>
```

CSS:

```
.transform {
  display: inline-block;
  width: 200px;
  height: 100px;
  background: teal;
  transition: transform 1s;
}

.origin1 {
  transform-origin: 0 0;
}

.origin2 {
  transform-origin: 100% 0;
}

.transform:hover {
  transform: rotate(30deg);
}
```

The default value for the transform-origin property is `50% 50%` which is the center of the element.

Read 2D Transforms online: <https://riptutorial.com/css/topic/938/2d-transforms>

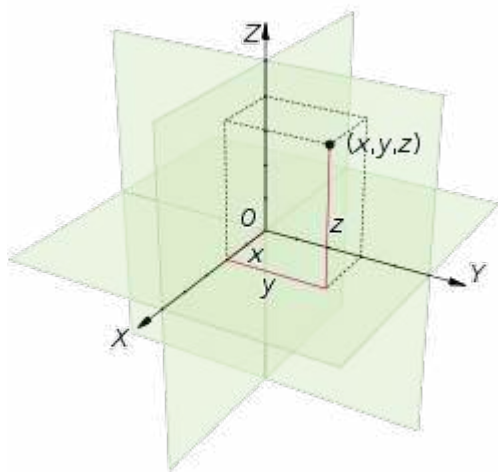
Chapter 3: 3D Transforms

Remarks

Coordinate system

3D transforms are made according to an (x, y, z) coordinate vector system in [Euclidean space](#).

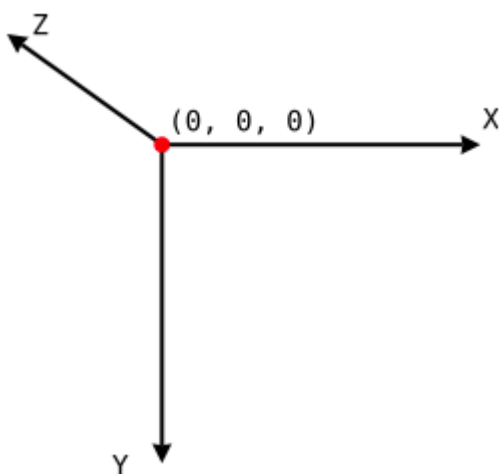
The following image shows an example of coordinates in Euclidean space:



In CSS,

- The x axis represents the horizontal (left and right)
- The y axis represents the vertical (up and down)
- The z axis represents the depth (forward and backward/closer and further)

The following image shows how these coordinates are translated in CSS:



Examples

3D cube

3D transforms can be use to create many 3D shapes. Here is a simple 3D CSS cube example:

HTML:

```
<div class="cube">
  <div class="cubeFace"></div>
  <div class="cubeFace face2"></div>
</div>
```

CSS:

```
body {
  perspective-origin: 50% 100%;
  perspective: 1500px;
  overflow: hidden;
}
.cube {
  position: relative;
  padding-bottom: 20%;
  transform-style: preserve-3d;
  transform-origin: 50% 100%;
  transform: rotateY(45deg) rotateX(0);
}
.cubeFace {
  position: absolute;
  top: 0;
  left: 40%;
  width: 20%;
  height: 100%;
  margin: 0 auto;
  transform-style: inherit;
  background: #C52329;
  box-shadow: inset 0 0 0 5px #333;
  transform-origin: 50% 50%;
  transform: rotateX(90deg);
  backface-visibility: hidden;
}
.face2 {
  transform-origin: 50% 50%;
  transform: rotatez(90deg) translateX(100%) rotateY(90deg);
}
.cubeFace:before, .cubeFace:after {
  content: '';
  position: absolute;
  width: 100%;
  height: 100%;
  transform-origin: 0 0;
  background: inherit;
  box-shadow: inherit;
  backface-visibility: inherit;
}
.cubeFace:before {
  top: 100%;
  left: 0;
  transform: rotateX(-90deg);
}
```

```
.cubeFace:after {  
  top: 0;  
  left: 100%;  
  transform: rotateY(90deg);  
}
```

[View this example](#)

Additional styling is added in the demo and a transform is applied on hover to view the 6 faces of the cube.

Should be noted that:

- 4 faces are made with pseudo elements
- [chained transforms](#) are applied

backface-visibility

The `backface-visibility` property relates to 3D transforms.

With 3D transforms and the `backface-visibility` property, you're able to rotate an element such that the original front of an element no longer faces the screen.

For example, this would flip an element away from the screen:

[JSFIDDLE](#)

```
<div class="flip">Loren ipsum</div>  
<div class="flip back">Lorem ipsum</div>
```

```
.flip {  
  -webkit-transform: rotateY(180deg);  
  -moz-transform: rotateY(180deg);  
  -ms-transform: rotateY(180deg);  
  -webkit-backface-visibility: visible;  
  -moz-backface-visibility: visible;  
  -ms-backface-visibility: visible;  
}  
  
.flip.back {  
  -webkit-backface-visibility: hidden;  
  -moz-backface-visibility: hidden;  
  -ms-backface-visibility: hidden;  
}
```

Firefox 10+ and IE 10+ support `backface-visibility` without a prefix. Opera, Chrome, Safari, iOS, and Android all need `-webkit-backface-visibility`.

It has 4 values:

1. **visible** (default) - the element will always be visible even when not facing the screen.
2. **hidden** - the element is not visible when not facing the screen.
3. **inherit** - the property will get its value from its parent element

4. **initial** - sets the property to its default, which is visible

Compass pointer or needle shape using 3D transforms

CSS

```
div.needle {  
  margin: 100px;  
  height: 150px;  
  width: 150px;  
  transform: rotateY(85deg) rotateZ(45deg);  
  /* presentational */  
  background-image: linear-gradient(to top left, #555 0%, #555 40%, #444 50%, #333 97%);  
  box-shadow: inset 6px 6px 22px 8px #272727;  
}
```

HTML

```
<div class='needle'></div>
```

In the above example, a needle or compass pointer shape is created using 3D transforms. Generally when we apply the `rotate` transform on an element, the rotation happens only in the Z-axis and at best we will end up with diamond shapes only. But when a `rotateY` transform is added on top of it, the element gets squeezed in the Y-axis and thus ends up looking like a needle. The more the rotation of the Y-axis the more squeezed the element looks.

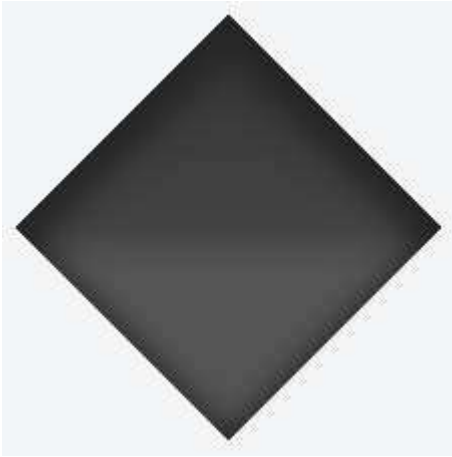
The output of the above example would be a needle resting on its tip. For creating a needle that is resting on its base, the rotation should be along the X-axis instead of along Y-axis. So the `transform` property's value would have to be something like `rotateX(85deg) rotateZ(45deg);`.

[This pen](#) uses a similar approach to create something that resembles the Safari logo or a compass dial.

Screenshot of element with no transform:



Screenshot of element with only 2D transform:



Screenshot of element with 3D transform:



3D text effect with shadow

HTML:

```
<div id="title">
  <h1 data-content="HOVER">HOVER</h1>
</div>
```

CSS:

```
*{margin:0;padding:0;}
html,body{height:100%;width:100%;overflow:hidden;background:#0099CC;}
#title{
  position:absolute;
  top:50%; left:50%;
  transform:translate(-50%,-50%);
  perspective-origin:50% 50%;
  perspective:300px;
}
h1{
  text-align:center;
  font-size:12vmin;
  font-family: 'Open Sans', sans-serif;
  color:rgba(0,0,0,0.8);
  line-height:1em;
```

```

transform: rotateY(50deg);
perspective: 150px;
perspective-origin: 0% 50%;
}
h1:after{
  content: attr(data-content);
  position: absolute;
  left: 0; top: 0;
  transform-origin: 50% 100%;
  transform: rotateX(-90deg);
  color: #0099CC;
}
#title:before{
  content: '';
  position: absolute;
  top: -150%; left: -25%;
  width: 180%; height: 328%;
  background: rgba(255, 255, 255, 0.7);
  transform-origin: 0 100%;
  transform: translateZ(-200px) rotate(40deg) skewX(35deg);
  border-radius: 0 0 100% 0;
}

```

[View example with additional hover effect](#)



In this example, the text is transformed to make it look like it is going into the screen away from the user.

The shadow is transformed accordingly so it follows the text. As it is made with a pseudo element and the `data` attribute, it inherits the transforms from its parent (the H1 tag).

The white "light" is made with a pseudo element on the `#title` element. It is skewed and uses `border-radius` for the rounded corner.

Read 3D Transforms online: <https://riptutorial.com/css/topic/2446/3d-transforms>

Chapter 4: Animations

Syntax

- `transition: <property> <duration> <timing-function> <delay>;`
- `@keyframes <identifier>`
- `[ [ from | to | <percentage> ] [, from | to | <percentage> ]* block ]*`

Parameters

Transition	
Parameter	Details
property	Either the CSS property to transition on, or <code>all</code> , which specifies all transition-able properties.
duration	Transition time, either in seconds or milliseconds.
timing-function	Specifies a function to define how intermediate values for properties are computed. Common values are <code>ease</code> , <code>linear</code> , and <code>step-end</code> . Check out the easing function cheat-sheet for more.
delay	Amount of time, in seconds or milliseconds, to wait before playing the animation.
@keyframes	
[from to <percentage>]	You can either specify a set time with a percentage value, or two percentage values, ie <code>10%</code> , <code>20%</code> , for a period of time where the keyframe's set attributes are set.
block	Any amount of CSS attributes for the keyframe.

Examples

Animations with the transition property

Useful for simple animations, the CSS `transition` property allows number-based CSS properties to animate between states.

Example

```
.Example{
  height: 100px;
  background: #fff;
}

.Example:hover{
  height: 120px;
  background: #ff0000;
}
```

[View Result](#)

By default, hovering over an element with the `.Example` class would immediately cause the element's height to jump to `120px` and its background color to red (`#ff0000`).

By adding the `transition` property, we can cause these changes to occur over time:

```
.Example{
  ...
  transition: all 400ms ease;
}
```

[View Result](#)

The `all` value applies the transition to all compatible (numbers-based) properties. Any compatible property name (such as `height` or `top`) can be substituted for this keyword.

`400ms` specifies the amount of time the transition takes. In this case, the element's change in height will take 400 milliseconds to complete.

Finally, the value `ease` is the animation function, which determines how the animation is played. `ease` means start slow, speed up, then end slow again. Other values are `linear`, `ease-out`, and `ease-in`.

Cross-Browser Compatibility

The `transition` property is generally well-supported across all major browsers, excepting IE 9. For earlier versions of Firefox and Webkit-based browsers, use vendor prefixes like so:

```
.Example{
  transition:          all 400ms ease;
  -moz-transition:     all 400ms ease;
  -webkit-transition:  all 400ms ease;
}
```

Note: The `transition` property can animate changes between any two numerical values, regardless of unit. It can also transition between units, such as `100px` to `50vh`. However, it cannot transition between a number and a default or automatic value, such as transitioning an element's height from `100px` to `auto`.

Increasing Animation Performance Using the `will-change` Attribute

When creating animations and other GPU-heavy actions, it's important to understand the `will-change` attribute.

Both CSS keyframes and the `transition` property use GPU acceleration. Performance is increased by offloading calculations to the device's GPU. This is done by creating paint layers (parts of the page that are individually rendered) that are offloaded to the GPU to be calculated. The `will-change` property tells the browser what will animate, allowing the browser to create smaller paint areas, thus increasing performance.

The `will-change` property accepts a comma-separated list of properties to be animated. For example, if you plan on transforming an object and changing its opacity, you would specify:

```
.Example{
  ...
  will-change: transform, opacity;
}
```

Note: Use `will-change` sparingly. Setting `will-change` for every element on a page can cause performance problems, as the browser may attempt to create paint layers for every element, significantly increasing the amount of processing done by the GPU.

Animations with keyframes

For multi-stage CSS animations, you can create CSS `@keyframes`. Keyframes allow you to define multiple animation points, called a keyframe, to define more complex animations.

Basic Example

In this example, we'll make a basic background animation that cycles between all colors.

```
@keyframes rainbow-background {
  0%      { background-color: #ff0000; }
  8.333%  { background-color: #ff8000; }
  16.667% { background-color: #ffff00; }
  25.000% { background-color: #80ff00; }
  33.333% { background-color: #00ff00; }
  41.667% { background-color: #00ff80; }
  50.000% { background-color: #00ffff; }
  58.333% { background-color: #0080ff; }
  66.667% { background-color: #0000ff; }
  75.000% { background-color: #8000ff; }
  83.333% { background-color: #ff00ff; }
```

```

    91.667%    { background-color: #ff0080; }
    100.00%   { background-color: #ff0000; }
}

.RainbowBackground {
    animation: rainbow-background 5s infinite;
}

```

View Result

There's a few different things to note here. First, the actual `@keyframes` syntax.

```
@keyframes rainbow-background{
```

This sets the name of the animation to `rainbow-background`.

```
0%    { background-color: #ff0000; }
```

This is the definition for a keyframe within the animation. The first part, the `0%` in the case, defines where the keyframe is during the animation. The `0%` implies it is 0% of the total animation time from the beginning.

The animation will automatically transition between keyframes. So, by setting the next background color at `8.333%`, the animation will smoothly take 8.333% of the time to transition between those keyframes.

```

.RainbowBackground {
    animation: rainbow-background 5s infinite;
}

```

This code attaches our animation to all elements which have the `.RainbowBackground` class.

The actual animation property takes the following arguments.

- **animation-name:** The name of our animation. In this case, `rainbow-background`
- **animation-duration:** How long the animation will take, in this case 5 seconds.
- **animation-iteration-count (Optional):** The number of times the animation will loop. In this case, the animation will go on indefinitely. By default, the animation will play once.
- **animation-delay (Optional):** Specifies how long to wait before the animation starts. It defaults to 0 seconds, and can take negative values. For example, `-2s` would start the animation 2 seconds into its loop.
- **animation-timing-function (Optional):** Specifies the speed curve of the animation. It defaults to `ease`, where the animation starts slow, gets faster and ends slow.

In this particular example, both the `0%` and `100%` keyframes specify `{ background-color: #ff0000; }`. Wherever two or more keyframes share a state, one may specify them in a single statement. In this case, the two `0%` and `100%` lines could be replaced with this single line:

```
0%, 100%    { background-color: #ff0000; }
```

Cross-browser compatibility

For older WebKit-based browsers, you'll need to use the vendor prefix on both the `@keyframes` declaration and the `animation` property, like so:

```
@-webkit-keyframes{}  
  
-webkit-animation: ...
```

Syntax Examples

Our first syntax example shows the animation shorthand property using all of the available properties/parameters:

```
animation: 3s          ease-in          1s    2          reverse    both  
paused    slidein;  
/*        duration | timing-function | delay | iteration-count | direction | fill-mode |  
play-state | name    */
```

Our second example is a little more simple, and shows that some properties can be omitted:

```
animation: 3s          linear          1s    slidein;  
/*        duration | timing-function | delay | name    */
```

Our third example shows the most minimal declaration. Note that the animation-name and animation-duration must be declared:

```
animation: 3s          slidein;  
/*        duration | name    */
```

It's also worth mentioning that when using the animation shorthand the order of the properties makes a difference. Obviously the browser may confuse your duration with your delay.

If brevity isn't your thing, you can also skip the shorthand property and write out each property individually:

```
animation-duration: 3s;  
animation-timing-function: ease-in;  
animation-delay: 1s;  
animation-iteration-count: 2;  
animation-direction: reverse;  
animation-fill-mode: both;  
animation-play-state: paused;  
animation-name: slidein;
```

Read Animations online: <https://riptutorial.com/css/topic/590/animations>

Chapter 5: Backgrounds

Introduction

With CSS you can set colors, gradients, and images as the background of an element.

It is possible to specify various combinations of images, colors, and gradients, and adjust the size, positioning, and repetition (among others) of these.

Syntax

- `background-color`: `color` | `transparent` | `initial` | `inherit`;
- `background-image`: `url` | `none` | `initial` | `inherit`;
- `background-position`: `value`;
- `background-size`: `<bg-size>` [`<bg-size>`]
- `<bg-size>`: `auto` | `length` | `cover` | `contain` | `initial` | `inherit`;
- `background-repeat`: `repeat` | `repeat-x` | `repeat-y` | `no-repeat` | `initial` | `inherit`;
- `background-origin`: `padding-box` | `border-box` | `content-box` | `initial` | `inherit`;
- `background-clip`: `border-box` | `padding-box` | `content-box` | `initial` | `inherit`;
- `background-attachment`: `scroll` | `fixed` | `local` | `initial` | `inherit`;
- `background`: `bg-color` `bg-image` `position` / `bg-size` `bg-repeat` `bg-origin` `bg-clip` `bg-attachment` `initial` | `inherit`;

Remarks

- CSS3 gradients will not work on versions of Internet Explorer less than 10.

Examples

Background Color

The `background-color` property sets the background color of an element using a color value or through keywords, such as `transparent`, `inherit` or `initial`.

- **transparent**, specifies that the background color should be transparent. This is default.
- **inherit**, inherits this property from its parent element.
- **initial**, sets this property to its default value.

This can be applied to all elements, and `::first-letter/::first-line` [pseudo-elements](#).

Colors in CSS can be specified by [different methods](#).

Color names

CSS

```
div {  
  background-color: red; /* red */  
}
```

HTML

```
<div>This will have a red background</div>
```

- The example used above is one of several ways that CSS has to represent a single color.
-

Hex color codes

Hex code is used to denote RGB components of a color in base-16 hexadecimal notation. `#ff0000`, for example, is bright red, where the red component of the color is 256 bits (ff) and the corresponding green and blue portions of the color is 0 (00).

If both values in each of the three RGB pairings (R, G, and B) are the same, then the color code can be shortened into three characters (the first digit of each pairing). `#ff0000` can be shortened to `#f00`, and `#ffffff` can be shortened to `#fff`.

Hex notation is case-insensitive.

```
body {  
  background-color: #de1205; /* red */  
}  
  
.main {  
  background-color: #00f; /* blue */  
}
```

RGB / RGBa

Another way to declare a color is to use RGB or RGBa.

RGB stands for Red, Green and Blue, and requires of three separate values between 0 and 255, put between brackets, that correspond with the decimal color values for respectively red, green and blue.

RGBA allows you to add an additional alpha parameter between 0.0 and 1.0 to define opacity.

```
header {  
  background-color: rgb(0, 0, 0); /* black */  
}
```

```
footer {
  background-color: rgba(0, 0, 0, 0.5); /* black with 50% opacity */
}
```

HSL / HSLa

Another way to declare a color is to use HSL or HSLa and is similar to RGB and RGBa.

HSL stands for hue, saturation, and lightness, and is also often called HLS:

- Hue is a degree on the color wheel (from 0 to 360).
- Saturation is a percentage between 0% and 100%.
- Lightness is also a percentage between 0% and 100%.

HSLa allows you to add an additional alpha parameter between 0.0 and 1.0 to define opacity.

```
li a {
  background-color: hsl(120, 100%, 50%); /* green */
}

#p1 {
  background-color: hsla(120, 100%, 50%, .3); /* green with 30% opacity */
}
```

Interaction with background-image

The following statements are all equivalent:

```
body {
  background: red;
  background-image: url(partiallytransparentimage.png);
}

body {
  background-color: red;
  background-image: url(partiallytransparentimage.png);
}

body {
  background-image: url(partiallytransparentimage.png);
  background-color: red;
}

body {
  background: red url(partiallytransparentimage.png);
}
```

They will all lead to the red color being shown underneath the image, where the parts of the image are transparent, or the image is not showing (perhaps as a result of `background-repeat`).

Note that the following is not equivalent:

```
body {  
  background-image: url(partiallytransparentimage.png);  
  background: red;  
}
```

Here, the value of `background` overrides your `background-image`.

For more info on the `background` property, see [Background Shorthand](#)

Background Image

The `background-image` property is used to specify a background image to be applied to all matched elements. By default, this image is tiled to cover the entire element, excluding margin.

```
.myClass {  
  background-image: url('/path/to/image.jpg');  
}
```

To use multiple images as `background-image`, define comma separated `url()`

```
.myClass {  
  background-image: url('/path/to/image.jpg'),  
                  url('/path/to/image2.jpg');  
}
```

The images will stack according to their order with the first declared image on top of the others and so on.

Value	Result
<code>url('/path/to/image.jpg')</code>	Specify background image's path(s) or an image resource specified with data URI schema (apostrophes can be omitted), separate multiples by comma
<code>none</code>	No background image
<code>initial</code>	Default value
<code>inherit</code>	Inherit parent's value

More CSS for Background Image

This following attributes are very useful and almost essential too.

```
background-size:    xpx ypx | x% y%;  
background-repeat:  no-repeat | repeat | repeat-x | repeat-y;  
background-position: left offset (px/%) right offset (px/%) | center center | left top | right bottom;
```

Background Gradients

Gradients are new image types, added in CSS3. As an image, gradients are set with the `background-image` property, or the `background` shorthand.

There are two types of gradient functions, linear and radial. Each type has a non-repeating variant and a repeating variant:

- `linear-gradient()`
- `repeating-linear-gradient()`
- `radial-gradient()`
- `repeating-radial-gradient()`

linear-gradient()

A `linear-gradient` has the following syntax

```
background: linear-gradient( <direction>?, <color-stop-1>, <color-stop-2>, ... );
```

Value	Meaning
<direction>	Could be an argument like <code>to top</code> , <code>to bottom</code> , <code>to right</code> or <code>to left</code> ; or an angle as <code>0deg</code> , <code>90deg</code> The angle starts from <code>to top</code> and rotates clockwise. Can be specified in deg , grad , rad , or turn . If omitted, the gradient flows from top to bottom
<color-stop-list>	List of colors, optionally followed each one by a percentage or length to display it at. For example, <code>yellow 10%</code> , <code>rgba(0,0,0,.5) 40px</code> , <code>#fff 100%</code> ...

For example, this creates a linear gradient that starts from the right and transitions from red to blue

```
.linear-gradient {  
  background: linear-gradient(to left, red, blue); /* you can also use 270deg */  
}
```

You can create a `diagonal` gradient by declaring both a horizontal and vertical starting position.

```
.diagonal-linear-gradient {  
  background: linear-gradient(to left top, red, yellow 10%);  
}
```

It is possible to specify any number of color stops in a gradient by separating them with commas. The following examples will create a gradient with 8 color stops

```
.linear-gradient-rainbow {  
  background: linear-gradient(to left, red, orange, yellow, green, blue, indigo, violet)  
}
```

radial-gradient()

```
.radial-gradient-simple {
  background: radial-gradient(red, blue);
}

.radial-gradient {
  background: radial-gradient(circle farthest-corner at top left, red, blue);
}
```

Value	Meaning
circle	Shape of gradient. Values are <code>circle</code> or <code>ellipse</code> , default is <code>ellipse</code> .
farthest-corner	Keywords describing how big the ending shape must be. Values are <code>closest-side</code> , <code>farthest-side</code> , <code>closest-corner</code> , <code>farthest-corner</code>
top left	Sets the position of the gradient center, in the same way as <code>background-position</code> .

Repeating gradients

Repeating gradient functions take the same arguments as the above examples, but tile the gradient across the background of the element.

```
.bullseye {
  background: repeating-radial-gradient(red, red 10%, white 10%, white 20%);
}

.warning {
  background: repeating-linear-gradient(-45deg, yellow, yellow 10%, black 10%, black 20% );
}
```

Value	Meaning
-45deg	Angle unit. The angle starts from to top and rotates clockwise. Can be specified in <code>deg</code> , <code>grad</code> , <code>rad</code> , or <code>turn</code> .
to left	Direction of gradient, default is <code>to bottom</code> . Syntax: <code>to [y-axis(top OR bottom)] [x-axis(left OR right)]</code> ie <code>to top right</code>
yellow 10%	Color, optionally followed by a percentage or length to display it at. Repeated two or more times.

Note that HEX, RGB, RGBA, HSL, and HSLa color codes may be used instead of color names. Color names were used for the sake of illustration. Also note that the radial-gradient syntax is much more complex than linear-gradient, and a simplified version is shown here. For a full

explanation and specs, see the [MDN Docs](#)

Background Shorthand

The `background` property can be used to set one or more background related properties:

Value	Description	CSS Ver.
<code>background-image</code>	Background image to use	1+
<code>background-color</code>	Background color to apply	1+
<code>background-position</code>	Background image's position	1+
<code>background-size</code>	Background image's size	3+
<code>background-repeat</code>	How to repeat background image	1+
<code>background-origin</code>	How the background is positioned (ignored when <code>background-attachment</code> is fixed)	3+
<code>background-clip</code>	How the background is painted relative to the <code>content-box</code> , <code>border-box</code> , or the <code>padding-box</code>	3+
<code>background-attachment</code>	How the background image behaves, whether it scrolls along with its containing block or has a fixed position within the viewport	1+
<code>initial</code>	Sets the property to value to default	3+
<code>inherit</code>	Inherits property value from parent	2+

The order of the values does not matter and every value is optional

Syntax

The syntax of the background shorthand declaration is:

```
background: [<background-image>] [<background-color>] [<background-position>]/[<background-size>] [<background-repeat>] [<background-origin>] [<background-clip>] [<background-attachment>] [<initial|inherit>];
```

Examples

```
background: red;
```

Simply setting a `background-color` with the `red` value.

```
background: border-box red;
```

Setting a `background-clip` to `border-box` and a `background-color` to `red`.

```
background: no-repeat center url("somepng.jpg");
```

Sets a `background-repeat` to `no-repeat`, `background-origin` to `center` and a `background-image` to an image.

```
background: url('pattern.png') green;
```

In this example, the `background-color` of the element would be set to `green` with `pattern.png`, if it is available, overlaid on the colour, repeating as often as necessary to fill the element. If `pattern.png` includes any transparency then the `green` colour will be visible behind it.

```
background: #000000 url("picture.png") top left / 600px auto no-repeat;
```

In this example we have a black background with an image 'picture.png' on top, the image does not repeat in either axis and is positioned in the top left corner. The `/` after the position is to be able to include the size of the background image which in this case is set as `600px` width and `auto` for the height. This example could work well with a feature image that can fade into a solid colour.

NOTE: Use of the shorthand `background` property resets all previously set background property values, even if a value is not given. If you wish only to modify a background property value previously set, use a longhand property instead.

Background Position

The `background-position` property is used to specify the starting position for a background image or gradient

```
.myClass {  
  background-image: url('path/to/image.jpg');  
  background-position: 50% 50%;  
}
```

The position is set using an **X** and **Y** co-ordinate and be set using any of the units used within CSS.

Unit	Description
<i>value%</i>	A percentage for the horizontal offset is relative to <i>(width of background positioning area - width of background image)</i> .
<i>value%</i>	A percentage for the vertical offset is relative to <i>(height of background positioning area - height of background image)</i>
	The size of the image is the size given by <code>background-size</code> .

Unit	Description
<i>valuepx</i> <i>valuepx</i>	Offsets background image by a length given in pixels relative to the top left of the background positioning area

Units in CSS can be specified by different methods (see [here](#)).

Longhand Background Position Properties

In addition to the shorthand property above, one can also use the longhand background properties `background-position-x` and `background-position-y`. These allow you to control the x or y positions separately.

NOTE: This is supported in all browsers except Firefox (versions 31-48) [2](#). Firefox 49, to be released September 2016, *will* support these properties. Until then, [there is a Firefox hack within this Stack Overflow answer](#).

Background Attachment

The `background-attachment` property sets whether a background image is fixed or scrolls with the rest of the page.

```
body {
  background-image: url('img.jpg');
  background-attachment: fixed;
}
```

Value	Description
scroll	The background scrolls along with the element. This is default.
fixed	The background is fixed with regard to the viewport.
local	The background scrolls along with the element's contents.
initial	Sets this property to its default value.
inherit	Inherits this property from its parent element.

Examples

background-attachment: scroll

The default behaviour, when the body is scrolled the background scrolls with it:

```
body {  
  background-image: url('image.jpg');  
  background-attachment: scroll;  
}
```

background-attachment: fixed

The background image will be fixed and will not move when the body is scrolled:

```
body {  
  background-image: url('image.jpg');  
  background-attachment: fixed;  
}
```

background-attachment: local

The background image of the div will scroll when the contents of the div is scrolled.

```
div {  
  background-image: url('image.jpg');  
  background-attachment: local;  
}
```

Background Repeat

The background-repeat property sets if/how a background image will be repeated.

By default, a background-image is repeated both vertically and horizontally.

```
div {  
  background-image: url("img.jpg");  
  background-repeat: repeat-y;  
}
```

Here's how a `background-repeat: repeat-y` looks like:



Background Color with Opacity

If you set `opacity` on an element it will affect all its child elements. To set an opacity just on the background of an element you will have to use RGBA colors. Following example will have a black background with 0.6 opacity.

```
/* Fallback for web browsers that don't support RGBA */
background-color: rgb(0, 0, 0);

/* RGBA with 0.6 opacity */
background-color: rgba(0, 0, 0, 0.6);

/* For IE 5.5 - 7*/
filter: progid:DXImageTransform.Microsoft.gradient(startColorstr=#99000000,
endColorstr=#99000000);

/* For IE 8*/
-ms-filter: "progid:DXImageTransform.Microsoft.gradient(startColorstr=#99000000,
endColorstr=#99000000)";
```

Multiple Background Image

In CSS3, we can stack multiple background in the same element.

```
#mydiv {
```

```
background-image: url(img_1.png), /* top image */
                  url(img_2.png), /* middle image */
                  url(img_3.png); /* bottom image */
background-position: right bottom,
                    left top,
                    right top;
background-repeat: no-repeat,
                  repeat,
                  no-repeat;
}
```

Images will be stacked atop one another with the first background on top and the last background in the back. `img_1` will be on top, the `img_2` and `img_3` is on bottom.

We can also use background shorthand property for this:

```
#mydiv {
  background: url(img_1.png) right bottom no-repeat,
             url(img_2.png) left top repeat,
             url(img_3.png) right top no-repeat;
}
```

We can also stack images and gradients:

```
#mydiv {
  background: url(image.png) right bottom no-repeat,
             linear-gradient(to bottom, #fff 0%, #000 100%);
}
```

- [Demo](#)

The background-origin property

The background-origin property specifies where the background image is positioned.

Note: If the `background-attachment` property is set to `fixed`, this property has no effect.

Default value: `padding-box`

Possible values:

- `padding-box` - The position is relative to the padding box
- `border-box` - The position is relative to the border box
- `content-box` - The position is relative to the content box
- `initial`
- `inherit`

CSS

```
.example {
  width: 300px;
  border: 20px solid black;
  padding: 50px;
}
```

```

    background: url(https://static.pexels.com/photos/6440/magazines-desk-work-workspace-
medium.jpg);
    background-repeat: no-repeat;
}

.example1 {}

.example2 { background-origin: border-box; }

.example3 { background-origin: content-box; }

```

HTML

```

<p>No background-origin (padding-box is default):</p>

<div class="example example1">
  <h2>Lorem Ipsum Dolor</h2>
  <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam nonummy nibh euismod
tincidunt ut laoreet dolore magna aliquam erat volutpat.</p>
  <p>Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis
nisl ut aliquip ex ea commodo consequat.</p>
</div>

<p>background-origin: border-box:</p>
<div class="example example2">
  <h2>Lorem Ipsum Dolor</h2>
  <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam nonummy nibh euismod
tincidunt ut laoreet dolore magna aliquam erat volutpat.</p>
  <p>Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis
nisl ut aliquip ex ea commodo consequat.</p>
</div>

<p>background-origin: content-box:</p>
<div class="example example3">
  <h2>Lorem Ipsum Dolor</h2>
  <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam nonummy nibh euismod
tincidunt ut laoreet dolore magna aliquam erat volutpat.</p>
  <p>Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis
nisl ut aliquip ex ea commodo consequat.</p>
</div>

```

Result:

No background-origin (padding-box is default):



background-origin: border-box:



background-origin: content-box:



is the default value. This allows the background to extend all the way to the outside edge of the element's border.

- `padding-box` clips the background at the outside edge of the element's padding and does not let it extend into the border;
- `content-box` clips the background at the edge of the content box.
- `inherit` applies the setting of the parent to the selected element.

CSS

```
.example {
  width: 300px;
  border: 20px solid black;
  padding: 50px;
  background: url(https://static.pexels.com/photos/6440/magazines-desk-work-workspace-medium.jpg);
  background-repeat: no-repeat;
}

.example1 {}

.example2 { background-origin: border-box; }

.example3 { background-origin: content-box; }
```

HTML

```
<p>No background-origin (padding-box is default):</p>

<div class="example example1">
  <h2>Lorem Ipsum Dolor</h2>
  <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.</p>
  <p>Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.</p>
</div>

<p>background-origin: border-box:</p>
<div class="example example2">
  <h2>Lorem Ipsum Dolor</h2>
  <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.</p>
  <p>Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.</p>
</div>

<p>background-origin: content-box:</p>
<div class="example example3">
  <h2>Lorem Ipsum Dolor</h2>
  <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.</p>
  <p>Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.</p>
</div>
```

Background Size

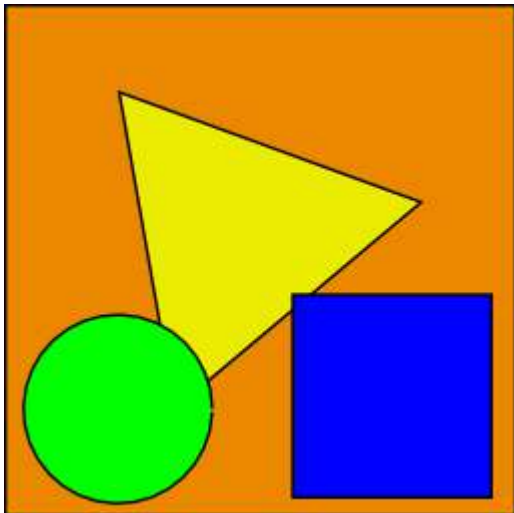
General overview

The `background-size` property enables one to control the scaling of the `background-image`. It takes up to two values, which determine the scale/size of the resulting image in vertical and horizontal direction. If the property is missing, it's deemed `auto` in both `width` and `height`.

`auto` will keep the image's aspect ratio, if it can be determined. The height is optional and can be considered `auto`. Therefore, on a 256 px × 256 px image, all the following `background-size` settings would yield an image with height and width of 50 px:

```
background-size: 50px;  
background-size: 50px auto; /* same as above */  
background-size: auto 50px;  
background-size: 50px 50px;
```

So if we started with the following picture (which has the mentioned size of 256 px × 256 px),



we'll end up with a 50 px × 50 px on the user's screen, contained in the background of our element:

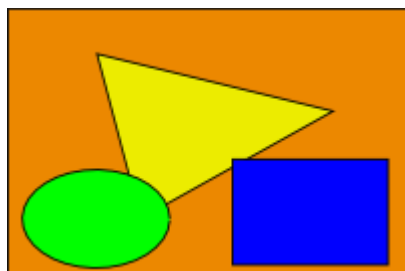


One can also use percentage values to scale the image with respect of the element. The following example would yield a 200 px × 133 px drawn image:

```
#withbackground {  
  background-image: url(to/some/background.png);  
  
  background-size: 100% 66%;  
  
  width: 200px;  
  height: 200px;  
  
  padding: 0;  
  margin: 0;
```



```
}
```



The behaviour depends on the `background-origin`.

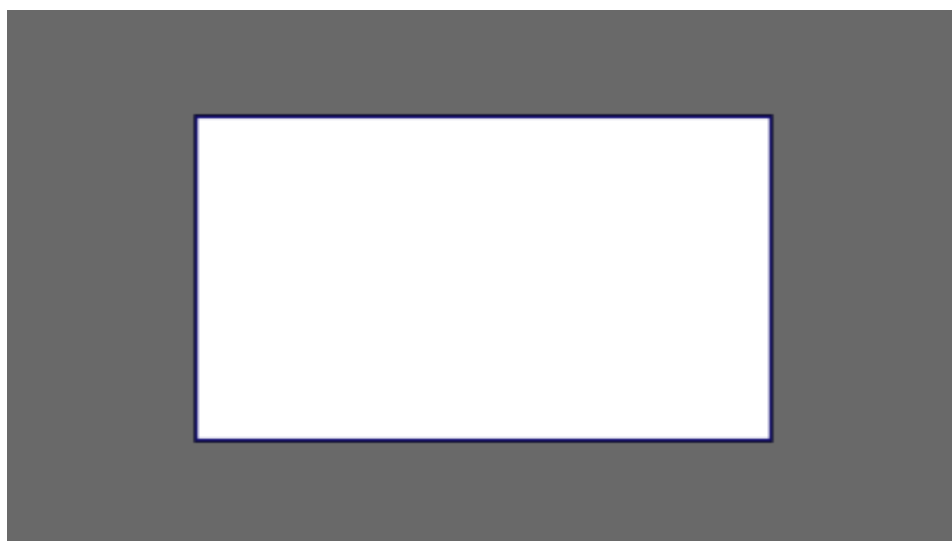
Keeping the aspect ratio

The last example in the previous section lost its original aspect ratio. The circle got into an ellipse, the square into a rectangle, the triangle into another triangle.

The length or percentage approach isn't flexible enough to keep the aspect ratio at all times. `auto` doesn't help, since you might not know which dimension of your element will be larger. However, to cover certain areas with an image (and correct aspect ratio) completely or to contain an image with correct aspect ratio completely in a background area, the values, `contain` and `cover` provide the additional functionality.

Eggsplanation for `contain` and `cover`

Sorry for the bad pun, but we're going to use a [picture of the day by Biswarup Ganguly](#) for demonstration. Let's say that this is your screen, and the gray area is outside of your visible screen. For demonstration, we're going to assume a 16 × 9 ratio.



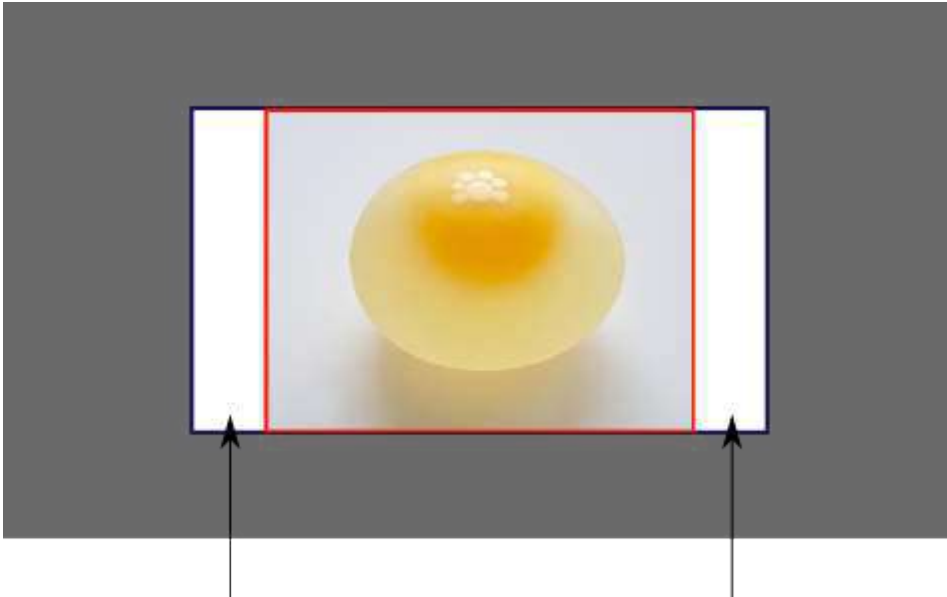
We want to use the aforementioned picture of the day as a background. However, we cropped the image to 4x3 for some reason. We could set the `background-size` property to some fixed length, but we will focus on `contain` and `cover`. Note that I also assume that we didn't mangle the width and/or height of `body`.

contain

```
contain
```

Scale the image, while preserving its intrinsic aspect ratio (if any), to the largest size such that both its width and its height can fit inside the background positioning area.

This makes sure that the background image is always completely contained in the background positioning area, however, there could be some empty space filled with your `background-color` in this case:

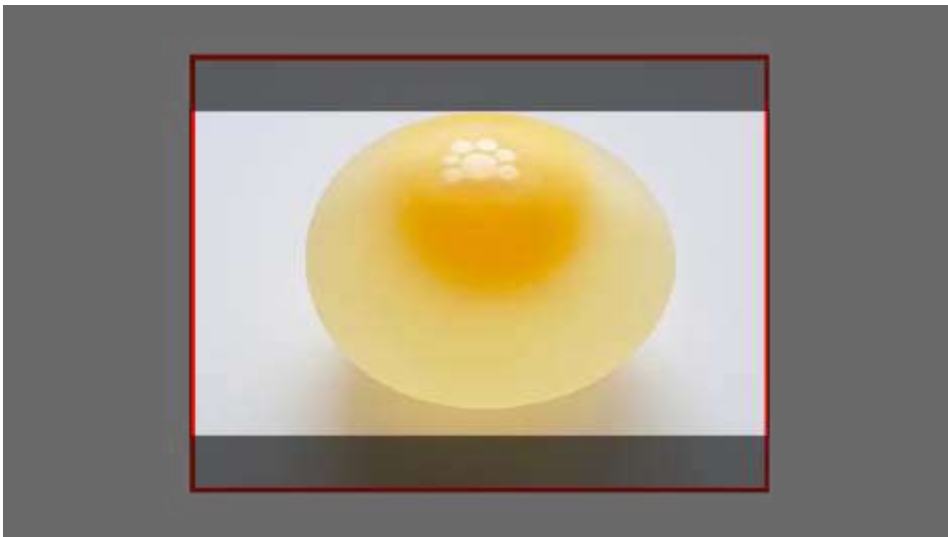


cover

```
cover
```

Scale the image, while preserving its intrinsic aspect ratio (if any), to the smallest size such that both its width and its height can completely cover the background positioning area.

This makes sure that the background image is covering everything. There will be no visible `background-color`, however depending on the screen's ratio a great part of your image could be cut off:



Demonstration with actual code

```
div > div {
  background-image: url(http://i.stack.imgur.com/r5CAq.jpg);
  background-repeat: no-repeat;
  background-position: center center;
  background-color: #ccc;
  border: 1px solid;
  width: 20em;
  height: 10em;
}
div.contain {
  background-size: contain;
}
div.cover {
  background-size: cover;
}
/*****
Additional styles for the explanation boxes
*****/

div > div {
  margin: 0 1ex 1ex 0;
  float: left;
}
div + div {
  clear: both;
  border-top: 1px dashed silver;
  padding-top: 1ex;
}
div > div::after {
  background-color: #000;
  color: #fefefe;
  margin: 1ex;
  padding: 1ex;
  opacity: 0.8;
  display: block;
  width: 10ex;
  font-size: 0.7em;
  content: attr(class);
}
```

```

<div>
  <div class="contain"></div>
  <p>Note the grey background. The image does not cover the whole region, but it's fully
<em>contained</em>.
  </p>
</div>
<div>
  <div class="cover"></div>
  <p>Note the ducks/geese at the bottom of the image. Most of the water is cut, as well as a
part of the sky. You don't see the complete image anymore, but neither do you see any
background color; the image <em>covers</em> all of the <code>&lt;div&gt;</code>.</p>
</div>

```



Note the grey background. The image does not cover the whole region, but it's fully *contained*.



Note the ducks/geese at the bottom of the image. Most of the water is cut, as well as a part of the sky. You don't see the complete image anymore, but neither do you see any background color; the image *covers* all of the <div>.

background-blend-mode Property

```

.my-div {
  width: 300px;
  height: 200px;
  background-size: 100%;
  background-repeat: no-repeat;
  background-image: linear-gradient(to right, black 0%,white 100%),
url('https://static.pexels.com/photos/54624/strawberry-fruit-red-sweet-54624-medium.jpeg');
  background-blend-mode:saturation;
}

```

```

<div class="my-div">Lorem ipsum</div>

```

See result here: <https://jsfiddle.net/MadalinaTn/y69d28Lb/>

CSS Syntax: background-blend-mode: normal | multiply | screen | overlay | darken | lighten | color-dodge | saturation | color | luminosity;

Read Backgrounds online: <https://riptutorial.com/css/topic/296/backgrounds>

Chapter 6: Block Formatting Contexts

Remarks

[A block formatting context is a part of a visual CSS rendering of a Web page. It is the region in which the layout of block boxes occurs and in which floats interact with each other.][1]

[1]: https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Block_formatting_context MDN

Examples

Using the overflow property with a value different to visible

```
img{
  float:left;
  width:100px;
  margin:0 10px;
}
.div1{
  background:#f1f1f1;
  /* does not create block formatting context */
}
.div2{
  background:#f1f1f1;
  overflow:hidden;
  /* creates block formatting context */
}
```

```

1 
2 <div class=div1>
3 <p>Lorem ipsum dolor sit amet, cum no paulo mollis pertinacia. Eam in velit graecis, sea mucus insolens ne. Amet doming at has, omnis errem an cum. Eu vim appareat persecuti, ea putant definitionem has, vis ea legendos expetenda. No eros graeci minimum nam, justo augue instructor usu ne. At ludus suscipit disputationi vel.</p>
4
5 <p>Ad case omnis nam, mutat deseruisse persequeris eos ad, in tollit debitis sea. Cu eos munere virtute vituperata. Exerci bonorum sed id, id nec tantas praesent complectitur. Vel cu legendos mediocritatem. Enim liberavisse ei sea.</p>
6 </div>
7
8 
9 <div class=div2>
10 <p>Lorem ipsum dolor sit amet, cum no paulo mollis pertinacia. Eam in velit graecis, sea mucus insolens ne. Amet doming at has, omnis errem an cum. Eu vim appareat persecuti, ea putant definitionem has, vis ea legendos expetenda. No eros graeci minimum nam, justo augue instructor usu ne. At ludus suscipit disputationi vel.</p>
11
12 <p>Ad case omnis nam, mutat deseruisse persequeris eos ad, in tollit debitis sea. Cu eos munere virtute vituperata. Exerci bonorum sed id, id nec tantas praesent complectitur. Vel cu legendos mediocritatem. Enim liberavisse ei sea.</p>
13 </div>

```

```

1 img{
2   float:left;
3   width:100px;
4   margin:0 10px;
5 }
6 .div1{
7   background:#f1f1f1;
8 }
9 .div2{
10  background:#f1f1f1;
11  overflow:hidden;
12  /* creates block formatting c
13 }

```



<https://jsfiddle.net/MadalinaTn/qkwwmu6m/2/>

Using the overflow property with a value different to visible (its default) will create a new block formatting context. This is technically necessary — if a float intersected with the scrolling element it would forcibly rewrap the content.

This example that show how a number of paragraphs will interact with a floated image is similar to [this example](#), on css-tricks.com.

2: <https://developer.mozilla.org/en-US/docs/Web/CSS/overflow> MDN

Read Block Formatting Contexts online: <https://riptutorial.com/css/topic/5069/block-formatting-contexts>

Chapter 7: Border

Syntax

- **border**
 - border: border-width border-style border-color | initial | inherit;
 - border-top: border-width border-style border-color | initial | inherit;
 - border-bottom: border-width border-style border-color | initial | inherit;
 - border-left: border-width border-style border-color | initial | inherit;
 - border-right: border-width border-style border-color | initial | inherit;
- **border-style**
 - border-style: 1-4 none | hidden | dotted | dashed | solid | double | groove | ridge | inset | outset | initial | inherit;
- **border-radius**
 - border-radius: 1-4 length | % / 1-4 length | % | initial | inherit;
 - border-top-left-radius: length | % [length | %] | initial | inherit;
 - border-top-right-radius: length | % [length | %] | initial | inherit;
 - border-bottom-left-radius: length | % [length | %] | initial | inherit;
 - border-bottom-right-radius: length | % [length | %] | initial | inherit;
- **border-image**
 - border-image: border-image-source border-image-slice [border-image-width [border-image-outset]] border-image-repeat
 - border-image-source: none | image;
 - border-image-slice: 1-4 number | percentage [fill]
 - border-image-repeat: 1-2 stretch | repeat | round | space
- **border-collapse**
 - border-collapse: separate | collapse | initial | inherit

Remarks

Related [properties](#):

- border
- border-bottom
- border-bottom-color
- border-bottom-left-radius
- border-bottom-right-radius
- border-bottom-style
- border-bottom-width
- border-color
- border-image
- border-image-outset
- border-image-repeat
- border-image-slice
- border-image-source
- border-image-width
- border-left
- border-left-color
- border-left-style
- border-left-width
- border-radius
- border-right
- border-right-color
- border-right-style
- border-right-width
- border-style
- border-top
- border-top-color

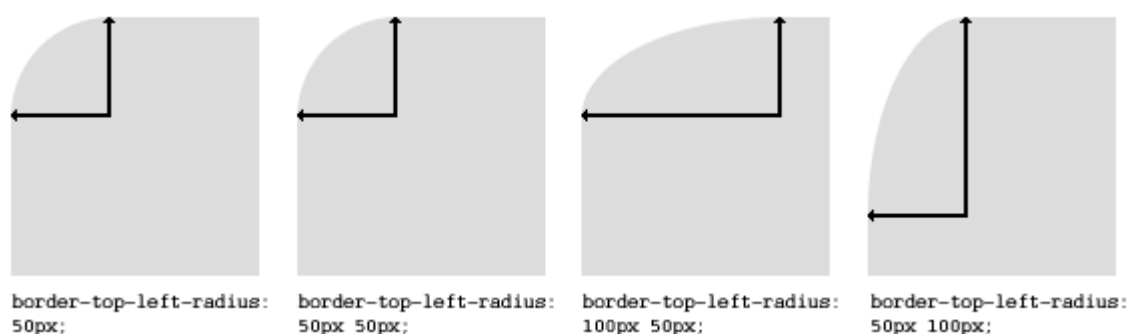
- border-top-left-radius
- border-top-right-radius
- border-top-style
- border-top-width
- border-width

Examples

border-radius

The border-radius property allows you to change the shape of the basic box model.

Every corner of an element can have up to two values, for the vertical and horizontal radius of that corner (for a maximum of 8 values).



The first set of values defines the horizontal radius. The optional second set of values, preceded by a '/' , defines the vertical radius. If only one set of values is supplied, it is used for both the vertical and horizontal radius.

```
border-radius: 10px 5% / 20px 25em 30px 35em;
```

The 10px is the horizontal radius of the top-left-and-bottom-right. And the 5% is the horizontal radius of the top-right-and-bottom-left. The other four values after '/' are the vertical radii for top-left, top-right, bottom-right and bottom-left.

As with many CSS properties, shorthands can be used for any or all possible values. You can therefore specify anything from one to eight values. The following shorthand allows you to set the horizontal and vertical radius of every corner to the same value:

HTML:

```
<div class='box'></div>
```

CSS:

```
.box {  
    width: 250px;  
    height: 250px;  
    background-color: black;  
    border-radius: 10px;  
}
```

Border-radius is most commonly used to convert box elements into circles. By setting the border-radius to half of the length of a square element, a circular element is created:

```
.circle {  
    width: 200px;  
    height: 200px;  
    border-radius: 100px;  
}
```

Because border-radius accepts percentages, it is common to use 50% to avoid manually calculating the border-radius value:

```
.circle {  
    width: 150px;  
    height: 150px;  
    border-radius: 50%;  
}
```

If the width and height properties are not equal, the resulting shape will be an oval rather than a circle.

Browser specific border-radius example:

```
-webkit-border-top-right-radius: 4px;  
-webkit-border-bottom-right-radius: 4px;  
-webkit-border-bottom-left-radius: 0;  
-webkit-border-top-left-radius: 0;  
-moz-border-radius-topright: 4px;  
-moz-border-radius-bottomright: 4px;  
-moz-border-radius-bottomleft: 0;  
-moz-border-radius-topleft: 0;  
border-top-right-radius: 4px;  
border-bottom-right-radius: 4px;  
border-bottom-left-radius: 0;  
border-top-left-radius: 0;
```

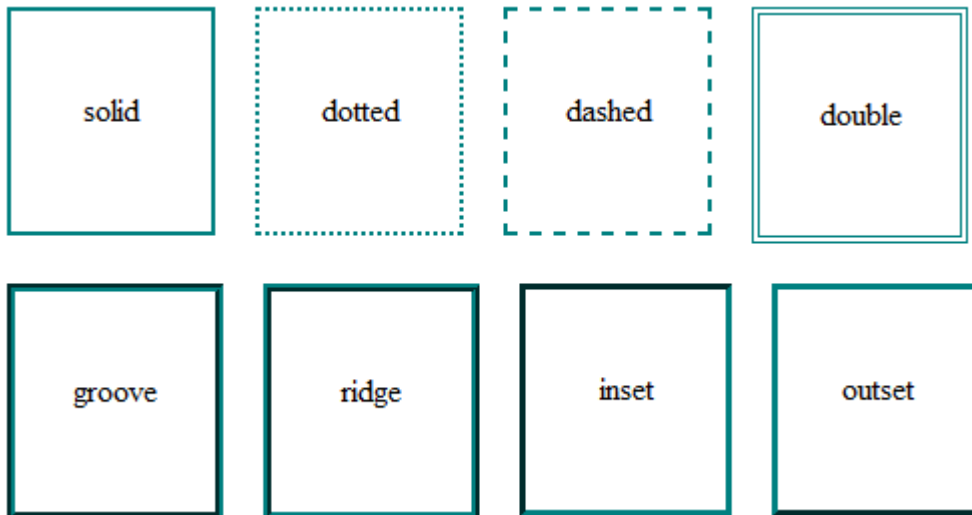
border-style

The `border-style` property sets the style of an element's border. This property can have from one to four values (for every side of the element one value.)

Examples:

```
border-style: dotted;
```

```
border-style: dotted solid double dashed;
```



`border-style` can also have the values `none` and `hidden`. They have the same effect, except `hidden` works for border conflict resolution for `<table>` elements. In a `<table>` with multiple borders, `none` has the lowest priority (meaning in a conflict, the border would show), and `hidden` has the highest priority (meaning in a conflict, the border would not show).

border (shorthands)

In most cases you want to define several border properties (`border-width`, `border-style` and `border-color`) for all sides of an element.

Instead of writing:

```
border-width: 1px;  
border-style: solid;  
border-color: #000;
```

You can simply write:

```
border: 1px solid #000;
```

These shorthands are also available for every side of an element: `border-top`, `border-left`, `border-right` and `border-bottom`. So you can do:

```
border-top: 2px double #aaaaaa;
```

border-image

With the `border-image` property you have the possibility to set an image to be used instead of normal border styles.

A `border-image` essentially consist of a

- `border-image-source`: The path to the image to be used
- `border-image-slice`: Specifies the offset that is used to divide the image into **nine regions** (four **corners**, four **edges** and a **middle**)
- `border-image-repeat`: Specifies how the images for the sides and the middle of the border image are scaled

Consider the following example whereas `border.png` is a image of 90x90 pixels:

```
border-image: url("border.png") 30 stretch;
```

The image will be split into nine regions with 30x30 pixels. The edges will be used as the corners of the border while the side will be used in between. If the element is higher / wider than 30px this part of the image will be **stretched**. The middle part of the image defaults to be transparent.

border-[left|right|top|bottom]

The `border-[left|right|top|bottom]` property is used to add a border to a specific side of an element.

For example if you wanted to add a border to the left side of an element, you could do:

```
#element {
    border-left: 1px solid black;
}
```

border-collapse

The `border-collapse` property applies only to `tables` (and elements displayed as `display: table` or `inline-table`) and sets whether the table borders are collapsed into a single border or detached as in standard HTML.

```
table {
    border-collapse: separate; /* default */
    border-spacing: 2px; /* Only works if border-collapse is separate */
}
```

Also see [Tables - border-collapse](#) documentation entry

Multiple Borders

Using outline:

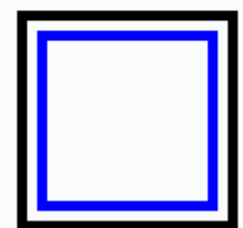
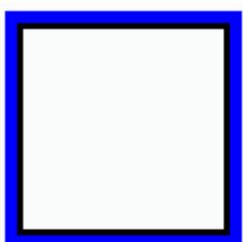
```
.div1{
    border: 3px solid black;
    outline: 6px solid blue;
    width: 100px;
    height: 100px;
    margin: 20px;
}
```

Using box-shadow:

```
.div2{  
  border: 5px solid green;  
  box-shadow: 0px 0px 0px 4px #000;  
  width: 100px;  
  height: 100px;  
  margin: 20px;  
}
```

Using a pseudo element:

```
.div3 {  
  position: relative;  
  border: 5px solid #000;  
  width: 100px;  
  height: 100px;  
  margin: 20px;  
}  
.div3:before {  
  content: " ";  
  position: absolute;  
  border: 5px solid blue;  
  z-index: -1;  
  top: 5px;  
  left: 5px;  
  right: 5px;  
  bottom: 5px;  
}
```



<http://jsfiddle.net/MadalinaTn/bvqpcohm/2/>

Creating a multi-colored border using border-image

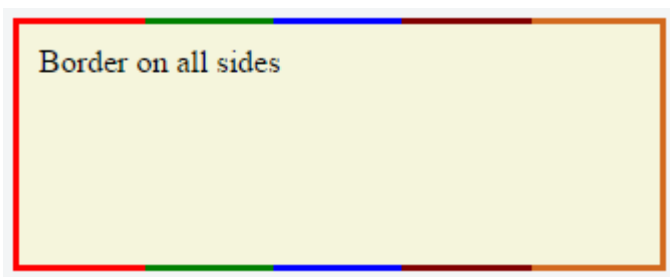
CSS

```
.bordered {  
  border-image: linear-gradient(to right, red 20%, green 20%, green 40%, blue 40%, blue 60%,  
    maroon 60%, maroon 80%, chocolate 80%); /* gradient with required colors */  
  border-image-slice: 1;  
}
```

HTML

```
<div class='bordered'>Border on all sides</div>
```

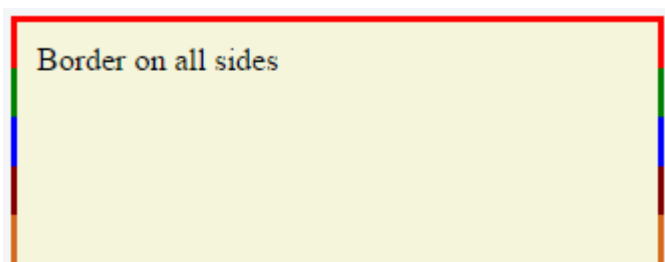
The above example would produce a border that comprises of 5 different colors. The colors are defined through a `linear-gradient` (you can find more information about gradients in the [docs](#)). You can find more information about `border-image-slice` property in the [border-image example](#) in same page.



(Note: Additional properties were added to the element for presentational purpose.)

You'd have noticed that the left border has only a single color (the start color of the gradient) while the right border also has only a single color (the gradient's end color). This is because of the way that border image property works. It is as though the gradient is applied to the entire box and then the colors are masked from the padding and content areas, thus making it look as though only the border has the gradient.

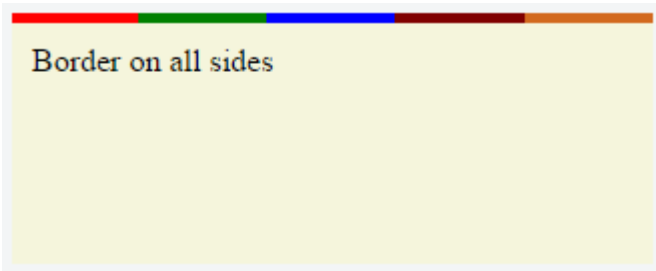
Which border(s) have a single color is dependant on the gradient definition. If the gradient is a `to right` gradient, the left border would be the start color of the gradient and right border would be the end color. If it was a `to bottom` gradient the top border would be the gradient's start color and bottom border would be end color. Below is the output of a `to bottom` 5 colored gradient.



If the border is required only on specific sides of the element then the `border-width` property can be

used just like with any other normal border. For example, adding the below code would produce a border only on the top of the element.

```
border-width: 5px 0px 0px 0px;
```



Note that, any element that has `border-image` property **won't respect the** `border-radius` (that is the border won't curve). This is based on the below statement in the spec:

A box's backgrounds, but not its border-image, are clipped to the appropriate curve (as determined by 'background-clip').

Read Border online: <https://riptutorial.com/css/topic/2160/border>

Chapter 8: box-shadow

Syntax

- `box-shadow: none|h-shadow v-shadow blur spread color [inset|initial|inherit];`

Parameters

Parameters	Details
inset	by default, the shadow is treated as a drop shadow. the inset keyword draws the shadow inside the frame/border.
offset-x	the horizontal distance
offset-y	the vertical distance
blur-radius	0 by default. value cannot be negative. the bigger the value, the bigger and lighter the shadow becomes.
spread-radius	0 by default. positive values will cause the shadow to expand. negative values will cause the shadow to shrink.
color	can be of various notations: a color keyword, hexadecimal, <code>rgb()</code> , <code>rgba()</code> , <code>hsl()</code> , <code>hsla()</code>

Remarks

Browser Support:

- Chrome 10.0
- IE 9.0
- Firefox 4.0 3.5 -moz
- Safari 5.1 3.1 -webkit-
- Opera 10.5

Examples

drop shadow

JSFiddle: <https://jsfiddle.net/UnsungHero97/80qpd7aL/>

HTML


```
<div class="box_shadow"></div>
```

CSS

```
.box_shadow {  
  -webkit-box-shadow: 0px 0px 10px -1px #444444;  
  -moz-box-shadow: 0px 0px 10px -1px #444444;  
  box-shadow: 0px 0px 10px -1px #444444;  
}
```

inner drop shadow

HTML

```
<div class="box_shadow"></div>
```

CSS

```
.box_shadow {  
  background-color: #1C90F3;  
  width: 200px;  
  height: 100px;  
  margin: 50px;  
  -webkit-box-shadow: inset 0px 0px 10px 0px #444444;  
  -moz-box-shadow: inset 0px 0px 10px 0px #444444;  
  box-shadow: inset 0px 0px 10px 0px #444444;  
}
```

Result:



JSFiddle: <https://jsfiddle.net/UnsungHero97/80qod7aL/1/>

bottom-only drop shadow using a pseudo-element

JSFiddle: <https://jsfiddle.net/UnsungHero97/80qod7aL/2/>

HTML

```
<div class="box_shadow"></div>
```

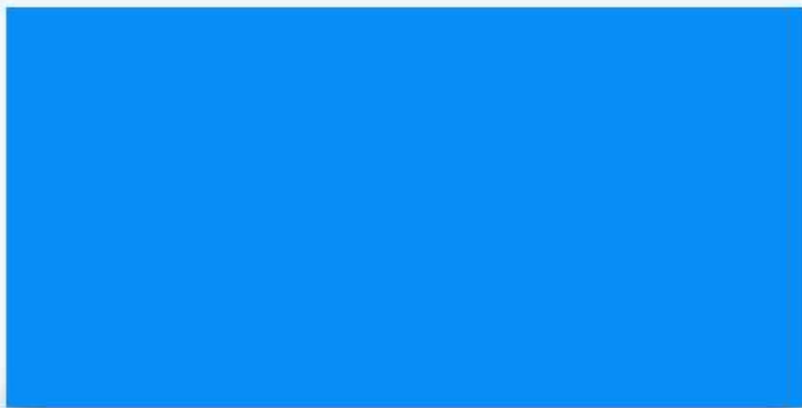
CSS

```

.box_shadow {
  background-color: #1C90F3;
  width: 200px;
  height: 100px;
  margin: 50px;
}

.box_shadow:after {
  content: "";
  width: 190px;
  height: 1px;
  margin-top: 98px;
  margin-left: 5px;
  display: block;
  position: absolute;
  z-index: -1;
  -webkit-box-shadow: 0px 0px 8px 2px #444444;
  -moz-box-shadow: 0px 0px 8px 2px #444444;
  box-shadow: 0px 0px 8px 2px #444444;
}

```



multiple shadows

JSFiddle: <https://jsfiddle.net/UnsungHero97/80qpd7aL/5/>

HTML

```
<div class="box_shadow"></div>
```

CSS

```

.box_shadow {
  width: 100px;
  height: 100px;
}

```

```
margin: 100px;  
box-shadow:  
  -52px -52px 0px 0px #f65314,  
  52px -52px 0px 0px #7cbb00,  
  -52px 52px 0px 0px #00a1f1,  
  52px 52px 0px 0px #ffbb00;  
}
```



Read box-shadow online: <https://riptutorial.com/css/topic/1746/box-shadow>

Chapter 9: Browser Support & Prefixes

Parameters

Prefix	Browser(s)
-webkit-	Google Chrome, Safari, newer versions of Opera 12 and up, Android, Blackberry and UC browsers
-moz-	Mozilla Firefox
-ms-	Internet Explorer, Edge
-o-, -xv-	Opera until version 12
-khtml-	Konquerer

Remarks

Vendor prefixes are used to allow preview support for new CSS functionality where the functionality is not yet recommended by the specification.

It is recommended that you do not use vendor prefixes in production environments. These prefixes exist to test new functionality that is not yet finalized, and behavior is inherently unexpected. Simply using prefixes does **not** grant browser support for old browsers as you cannot guarantee the feature hasn't changed over time to perform differently, and it *could* still be broken in those old browsers you claim to support.

If supporting older browsers is important, you should instead consider using JavaScript or other solutions to imitate the effects and truly guarantee support for old browsers.

Browsers will use their prefixes and ignore the properties they don't understand.

NOTE: Prefixes should always appear before the official, unprefixed syntax. Otherwise they would be overwritten with the prefixed properties, which can be another implementation in the end.

If a browser supports both an unprefixed and prefixed version of a property, the most recent property to be declared will take precedence.

Examples

Transitions

```
div {  
  -webkit-transition: all 4s ease;  
  -moz-transition: all 4s ease;  
  -o-transition: all 4s ease;  
  transition: all 4s ease;  
}
```

Transform

```
div {  
  -webkit-transform: rotate(45deg);  
  -moz-transform: rotate(45deg);  
  -ms-transform: rotate(45deg);  
  -o-transform: rotate(45deg);  
  transform: rotate(45deg);  
}
```

Read Browser Support & Prefixes online: <https://riptutorial.com/css/topic/1138/browser-support---prefixes>

Chapter 10: Cascading and Specificity

Remarks

CSS specificity intends to promote code conciseness by allowing an author to define some general formatting rules for a broad set of elements, and then to override them for a certain subset.

Examples

Cascading

Cascading and specificity are used together to determine the final value of a CSS styling property. They also define the mechanisms for resolving conflicts in CSS rule sets.

CSS Loading order

Styles are read from the following sources, in this order:

1. User Agent stylesheet (The styles supplied by the browser vendor)
2. User stylesheet (The additional styling a user has set on his/her browser)
3. Author stylesheet (Author here means the creator of the webpage/website)
 - Maybe one or more `.css` files
 - In the `<style>` element of the HTML document
4. Inline styles (In the `style` attribute on an HTML element)

The browser will lookup the corresponding style(s) when rendering an element.

How are conflicts resolved?

When only one CSS rule set is trying to set a style for an element, then there is no conflict, and that rule set is used.

When multiple rule sets are found with conflicting settings, first the Specificity rules, and then the Cascading rules are used to determine what style to use.

Example 1 - Specificity rules

```
.mystyle { color: blue; } /* specificity: 0, 0, 1, 0 */
div { color: red; }      /* specificity: 0, 0, 0, 1 */
```

```
<div class="mystyle">Hello World</div>
```

What color will the text be? (hover to see the answer)

blue

First the specificity rules are applied, and the one with the highest specificity "wins".

Example 2 - Cascade rules with identical selectors

External css file

```
.class {  
  background: #FFF;  
}
```

Internal css (in HTML file)

```
<style>  
.class {  
  background: #000;  
}  
</style>
```

In this case, where you have identical selectors, the cascade kicks in, and determines that the last one loaded "wins".

Example 3 - Cascade rules after Specificity rules

```
body > .mystyle { background-color: blue; } /* specificity: 0, 0, 1, 1 */  
.otherstyle > div { background-color: red; } /* specificity: 0, 0, 1, 1 */
```

```
<body class="otherstyle">  
  <div class="mystyle">Hello World</div>  
</body>
```

What color will the background be?

red

After applying the specificity rules, there's still a conflict between blue and red, so the cascading rules are applied on top of the specificity rules. Cascading looks at the load order of the rules, whether inside the same `.css` file or in the collection of style sources. The last one loaded overrides any earlier ones. In this case, the `.otherstyle > div` rule "wins".

A final note

- Selector specificity always take precedence.
- Stylesheet order break ties.
- Inline styles trump everything.

The !important declaration

The `!important` declaration is used to override the usual specificity in a style sheet by giving a higher priority to a rule. Its usage is: `property : value !important;`

```
#mydiv {
    font-weight: bold !important;    /* This property won't be overridden
                                     by the rule below */
}

#outerdiv #mydiv {
    font-weight: normal;             /* #mydiv font-weight won't be set to normal
                                     even if it has a higher specificity because
                                     of the !important declaration above */
}
```

Avoiding the usage of `!important` is strongly recommended (unless absolutely necessary), because it will disturb the natural flow of css rules which can bring uncertainty in your style sheet. Also it is important to note that when multiple `!important` declarations are applied to the same rule on a certain element, the one with the higher specificity will be the one applied.

Here are some examples where using `!important` declaration can be justified:

- If your rules shouldn't be overridden by any inline style of the element which is written inside `style` attribute of the html element.
- To give the user more control over the web accessibility, like increasing or decreasing size of the font-size, by overriding the author style using `!important`.
- For testing and debugging using inspect element.

See also:

- [W3C - 6 Assigning property values, Cascading, and Inheritance -- 6.4.2 !important rules](#)

Calculating Selector Specificity

Each individual CSS Selector has its own specificity value. Every selector in a sequence increases the sequence's overall specificity. Selectors fall into one of three different specificity groups: *A*, *B* and *c*. When multiple selector sequences select a given element, the browser uses the styles applied by the sequence with the highest overall specificity.

Group	Comprised of	Examples
<i>A</i>	id selectors	#foo
<i>B</i>	class selectors attribute selectors pseudo-classes	.bar [title], [colspan="2"] :hover, :nth-child(2)
<i>c</i>	type selectors	div, li

Group	Comprised of	Examples
	pseudo-elements	::before, ::first-letter

Group *A* is the most specific, followed by Group *B*, then finally Group *C*.

The universal selector (*) and combinators (like > and ~) have no specificity.

Example 1: Specificity of various selector sequences

```
#foo #baz {}          /* a=2, b=0, c=0 */
#foo.bar {}          /* a=1, b=1, c=0 */
#foo {}              /* a=1, b=0, c=0 */
.bar:hover {}        /* a=0, b=2, c=0 */
div.bar {}           /* a=0, b=1, c=1 */
:hover {}            /* a=0, b=1, c=0 */
[title] {}           /* a=0, b=1, c=0 */
.bar {}              /* a=0, b=1, c=0 */
div ul + li {}       /* a=0, b=0, c=3 */
p::after {}          /* a=0, b=0, c=2 */
*::before {}         /* a=0, b=0, c=1 */
::before {}          /* a=0, b=0, c=1 */
div {}               /* a=0, b=0, c=1 */
* {}                 /* a=0, b=0, c=0 */
```

Example 2: How specificity is used by the browser

Imagine the following CSS implementation:

```
#foo {
  color: blue;
}

.bar {
  color: red;
  background: black;
}
```

Here we have an ID selector which declares `color` as *blue*, and a class selector which declares `color` as *red* and `background` as *black*.

An element with an ID of `#foo` and a class of `.bar` will be selected by both declarations. ID

selectors have a Group *A* specificity and class selectors have a Group *B* specificity. An ID selector outweighs any number of class selectors. Because of this, `color:blue;` from the `#foo` selector and the `background:black;` from the `.bar` selector will be applied to the element. The higher specificity of the ID selector will cause the browser to ignore the `.bar` selector's `color` declaration.

Now imagine a different CSS implementation:

```
.bar {  
  color: red;  
  background: black;  
}  
  
.baz {  
  background: white;  
}
```

Here we have two class selectors; one of which declares `color` as *red* and `background` as *black*, and the other declares `background` as *white*.

An element with both the `.bar` and `.baz` classes will be affected by both of these declarations, however the problem we have now is that both `.bar` and `.baz` have an identical Group *B* specificity. The cascading nature of CSS resolves this for us: as `.baz` is defined *after* `.bar`, our element ends up with the *red* `color` from `.bar` but the *white* `background` from `.baz`.

Example 3: How to manipulate specificity

The last snippet from Example 2 above can be manipulated to ensure our `.bar` class selector's `color` declaration is used instead of that of the `.baz` class selector.

```
.bar {}          /* a=0, b=1, c=0 */  
.baz {}          /* a=0, b=1, c=0 */
```

The most common way to achieve this would be to find out what other selectors can be applied to the `.bar` selector sequence. For example, if the `.bar` class was only ever applied to `span` elements, we could modify the `.bar` selector to `span.bar`. This would give it a new Group *C* specificity, which would override the `.baz` selector's lack thereof:

```
span.bar {}      /* a=0, b=1, c=1 */  
.baz {}          /* a=0, b=1, c=0 */
```

However it may not always be possible to find another common selector which is shared between any element which uses the `.bar` class. Because of this, CSS allows us to duplicate selectors to increase specificity. Instead of just `.bar`, we can use `.bar.bar` instead (See [The grammar of Selectors, W3C Recommendation](#)). This still selects any element with a class of `.bar`, but now has double the Group *B* specificity:

```
.bar.bar {}      /* a=0, b=2, c=0 */  
.baz {}          /* a=0, b=1, c=0 */
```

`!important` and inline style declarations

The `!important` flag on a style declaration and styles declared by the `HTML style` attribute are considered to have a greater specificity than any selector. If these exist, the style declaration they affect will overrule other declarations regardless of their specificity. That is, unless you have more than one declaration that contains an `!important` flag for the same property that apply to the same element. Then, normal specificity rules will apply to those properties in reference to each other.

Because they completely override specificity, the use of `!important` is frowned upon in most use cases. One should use it as little as possible. To keep CSS code efficient and maintainable in the long run, it's almost always better to increase the specificity of the surrounding selector than to use `!important`.

One of those rare exceptions where `!important` is not frowned upon, is when implementing generic helper classes like a `.hidden` or `.background-yellow` class that are supposed to always override one or more properties wherever they are encountered. And even then, you need to know what you're doing. The last thing you want, when writing maintainable CSS, is to have `!important` flags throughout your CSS.

A final note

A common misconception about CSS specificity is that the Group *A*, *B* and *c* values should be combined with each other ($a=1, b=5, c=1 \Rightarrow 151$). This is **not** the case. If this were the case, having 20 of a Group *B* or *c* selector would be enough to override a single Group *A* or *B* selector respectively. The three groups should be regarded as individual levels of specificity. Specificity cannot be represented by a single value.

When creating your CSS style sheet, you should maintain the lowest specificity as possible. If you need to make the specificity a little higher to overwrite another method, make it higher but as low as possible to make it higher. You shouldn't need to have a selector like this:

```
body.page header.container nav div#main-nav li a {}
```

This makes future changes harder and pollutes that css page.

You can calculate the specificity of your selector [here](#)

More complex specificity example

```
div {
  font-size: 7px;
  border: 3px dotted pink;
  background-color: yellow;
  color: purple;
}

body.mystyle > div.myotherstyle {
  font-size: 11px;
```

```

    background-color: green;
}

#elmnt1 {
    font-size: 24px;
    border-color: red;
}

.mystyle .myotherstyle {
    font-size: 16px;
    background-color: black;
    color: red;
}

```

```

<body class="mystyle">
  <div id="elmnt1" class="myotherstyle">
    Hello, world!
  </div>
</body>

```

What borders, colors, and font-sizes will the text be?

font-size:

font-size: 24;;, since `#elmnt1` rule set has the highest specificity for the `<div>` in question, every property here is set.

border:

border: 3px dotted red;. The border-color `red` is taken from `#elmnt1` rule set, since it has the highest specificity. The other properties of the border, border-thickness, and border-style are from the `div` rule set.

background-color:

background-color: green;. The background-color is set in the `div`, `body.mystyle` > `div.myotherstyle`, and `.mystyle .myotherstyle` rule sets. The specificities are (0, 0, 1) vs. (0, 2, 2) vs. (0, 2, 0), so the middle one "wins".

color:

color: red;. The color is set in both the `div` and `.mystyle .myotherstyle` rule sets. The latter has the higher specificity of (0, 2, 0) and "wins".

Read Cascading and Specificity online: <https://riptutorial.com/css/topic/450/cascading-and-specificity>

Chapter 11: Centering

Examples

Using CSS transform

[CSS transforms](#) are based on the size of the elements so if you don't know how tall or wide your element is, you can position it absolutely 50% from the top and left of a relative container and translate it by 50% left and upwards to center it vertically and horizontally.

Keep in mind that with this technique, the element could end being rendered at a non-integer pixel boundary, making it look blurry. See [this answer in SO](#) for a workaround.

HTML

```
<div class="container">
  <div class="element"></div>
</div>
```

CSS

```
.container {
  position: relative;
}

.element {
  position: absolute;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
}
```

[View example in JSFiddle](#)

CROSS BROWSER COMPATIBILITY

The transform property needs prefixes to be supported by older browsers. Prefixes are needed for Chrome<=35, Safari<=8, Opera<=22, Android Browser<=4.4.4, and IE9. CSS transforms are not supported by IE8 and older versions.

Here is a common transform declaration for the previous example:

```
-webkit-transform: translate(-50%, -50%); /* Chrome, Safari, Opera, Android */
-ms-transform: translate(-50%, -50%); /* IE 9 */
transform: translate(-50%, -50%);
```

For more information see [canluse](#).

MORE INFORMATION

- The element is being positioned according to the first non-static parent (`position: relative`, `absolute`, or `fixed`). Explore more in this [fiddle](#) and this [documentation topic](#).
- For horizontal-only centering, use `left: 50%` and `transform: translateX(-50%)`. The same goes for vertical-only centering: center with `top: 50%` and `transform: translateY(-50%)`.
- Using a non-static width/height elements with this method of centering can cause the centered element to appear squished. This mostly happens with elements containing text, and can be fixed by adding: `margin-right: -50%;` and `margin-bottom: -50%;`. View this [fiddle](#) for more information.

Using Flexbox

HTML:

```
<div class="container">
  
</div>
```

CSS:

```
html, body, .container {
  height: 100%;
}
.container {
  display: flex;
  justify-content: center; /* horizontal center */
}
img {
  align-self: center; /* vertical center */
}
```

[View Result](#)

HTML:

```

```

CSS:

```
html, body {
  height: 100%;
}
body {
  display: flex;
  justify-content: center; /* horizontal center */
  align-items: center;    /* vertical center */
}
```

[View Result](#)

See [Dynamic Vertical and Horizontal Centering](#) under the [Flexbox](#) documentation for more details on flexbox and what the styles mean.

Browser Support

Flexbox is supported by all major browsers, [except IE versions before 10](#).

Some recent browser versions, such as Safari 8 and IE10, require [vendor prefixes](#).

For a quick way to generate prefixes there is [Autoprefixer](#), a third-party tool.

For older browsers (like IE 8 & 9) a [Polyfill is available](#).

For a more detailed look at flexbox browser support, see [this answer](#).

Using position: absolute

Working in old browsers (IE >= 8)

Automatic margins, paired with values of zero for the `left` and `right` or `top` and `bottom` offsets, will center an absolutely positioned elements within its parent.

[View Result](#)

HTML

```
<div class="parent">
  
</div>
```

CSS

```
.parent {
  position: relative;
  height: 500px;
}

.center {
  position: absolute;
  margin: auto;
  top: 0;
  right: 0;
  bottom: 0;
  left: 0;
}
```

Elements that don't have their own implicit width and height like images do, will need those values defined.

Other resources: [Absolute Centering in CSS](#)

Ghost element technique (Michał Czernew's hack)

This technique works even when the container's dimensions are unknown.

Set up a "ghost" element inside the container to be centered that is 100% height, then use `vertical-align: middle` on both that and the element to be centered.

CSS

```
/* This parent can be any width and height */
.block {
    text-align: center;

    /* May want to do this if there is risk the container may be narrower than the element
    inside */
    white-space: nowrap;
}

/* The ghost element */
.block:before {
    content: '';
    display: inline-block;
    height: 100%;
    vertical-align: middle;

    /* There is a gap between ghost element and .centered,
    caused by space character rendered. Could be eliminated by
    nudging .centered (nudge distance depends on font family),
    or by zeroing font-size in .parent and resetting it back
    (probably to 1rem) in .centered. */
    margin-right: -0.25em;
}

/* The element to be centered, can also be of any width and height */
.centered {
    display: inline-block;
    vertical-align: middle;
    width: 300px;
    white-space: normal; /* Resetting inherited nowrap behavior */
}
```

HTML

```
<div class="block">
  <div class="centered"></div>
</div>
```

Using text-align

The most common and easiest type of centering is that of lines of text in an element. CSS has the rule `text-align: center` for this purpose:

HTML


```
<p>Lorem ipsum</p>
```

CSS

```
p {  
  text-align: center;  
}
```

This does not work for centering entire block elements. `text-align` controls only alignment of inline content like text in its parent block element.

See more about `text-align` in [Typography](#) section.

Centering in relation to another item

We will see how to center content based on the height of a near element.

Compatibility: IE8+, all other modern browsers.

HTML

```
<div class="content">  
  <div class="position-container">  
    <div class="thumb">  
        
    </div>  
    <div class="details">  
      <p class="banner-title">text 1</p>  
      <p class="banner-text">content content content content content content content  
content content content content content content</p>  
      <button class="btn">button</button>  
    </div>  
  </div>  
</div>
```

CSS

```
.content * {  
  box-sizing: border-box;  
}  
.content .position-container {  
  display: table;  
}  
.content .details {  
  display: table-cell;  
  vertical-align: middle;  
  width: 33.333333%;  
  padding: 30px;  
  font-size: 17px;  
  text-align: center;  
}  
.content .thumb {  
  width: 100%;  
}  
.content .thumb img {
```

```
width: 100%;
}
```

Link to [JSFiddle](#)

The main points are the 3 `.thumb`, `.details` and `.position-container` containers:

- The `.position-container` must have `display: table`.
- The `.details` must have the real width set `width: ...` and `display: table-cell, vertical-align: middle`.
- The `.thumb` must have `width: 100%` if you want that it will take all the remaining space and it will be influenced by the `.details` width.
- The image (if you have an image) inside `.thumb` should have `width: 100%`, but it is not necessary if you have correct proportions.

Vertical align anything with 3 lines of code

Supported by IE11+

[View Result](#)

Use these 3 lines to vertical align practically everything. Just make sure the div/image you apply the code to has a parent with a height.

CSS

```
div.vertical {
  position: relative;
  top: 50%;
  transform: translateY(-50%);
}
```

HTML

```
<div class="vertical">Vertical aligned text!</div>
```

Vertically align an image inside div

HTML

```
<div class="wrap">
  
</div>
```

CSS

```
.wrap {
```

```

    height: 50px; /* max image height */
    width: 100px;
    border: 1px solid blue;
    text-align: center;
}
.wrap:before {
    content: "";
    display: inline-block;
    height: 100%;
    vertical-align: middle;
    width: 1px;
}

img {
    vertical-align: middle;
}

```

Horizontal and Vertical centering using table layout

One could easily center a child element using `table` display property.

HTML

```

<div class="wrapper">
  <div class="parent">
    <div class="child"></div>
  </div>
</div>

```

CSS

```

.wrapper {
    display: table;
    vertical-align: center;
    width: 200px;
    height: 200px;
    background-color: #9e9e9e;
}
.parent {
    display: table-cell;
    vertical-align: middle;
    text-align: center;
}
.child {
    display: inline-block;
    vertical-align: middle;
    text-align: center;
    width: 100px;
    height: 100px;
    background-color: teal;
}

```

Using calc()

The `calc()` function is the part of a new syntax in CSS3 in which you can calculate (mathematically) what size/position your element occupies by using a variety of values like pixels, percentages, etc.

Note:- Whenever you use this function, always take care of the space between two values

```
calc(100% - 80px).
```

CSS

```
.center {
  position: absolute;
  height: 50px;
  width: 50px;
  background: red;
  top: calc(50% - 50px / 2); /* height divided by 2*/
  left: calc(50% - 50px / 2); /* width divided by 2*/
}
```

HTML

```
<div class="center"></div>
```

Vertically align dynamic height elements

Applying css intuitively doesn't produce the desired results because

- `vertical-align:middle` *isn't applicable to block-level elements*
- `margin-top:auto` and `margin-bottom:auto` *used values would compute as zero*
- `margin-top:-50%` *percentage-based margin values are calculated relative to the width of containing block*

For widest browser support, a workaround with helper elements:

HTML

```
<div class="vcenter--container">
  <div class="vcenter--helper">
    <div class="vcenter--content">
      <!--stuff-->
    </div>
  </div>
</div>
```

CSS

```
.vcenter--container {
  display: table;
  height: 100%;
  position: absolute;
  overflow: hidden;
  width: 100%;
}
.vcenter--helper {
  display: table-cell;
  vertical-align: middle;
}
.vcenter--content {
```

```
margin: 0 auto;
width: 200px;
}
```

[jsfiddle](#) from [original question](#). This approach

- works with dynamic height elements
- respects content flow
- is supported by legacy browsers

Using line-height

You can also use `line-height` to center vertically a single line of text inside a container :

CSS

```
div {
  height: 200px;
  line-height: 200px;
}
```

That's quite ugly, but can be useful inside an `<input />` element. The `line-height` property works only when the text to be centered spans a single line. If the text wraps into multiple lines, the resulting output won't be centered.

Centering vertically and horizontally without worrying about height or width

The following technique allows you to add your content to an HTML element and center it both horizontally and vertically **without worrying about its height or width**.

The outer container

- should have `display: table;`

The inner container

- should have `display: table-cell;`
- should have `vertical-align: middle;`
- should have `text-align: center;`

The content box

- should have `display: inline-block;`
- should re-adjust the horizontal text-alignment to eg. `text-align: left;` **or** `text-align: right;`, unless you want text to be centered

Demo

HTML

```
<div class="outer-container">
  <div class="inner-container">
    <div class="centered-content">
      You can put anything here!
    </div>
  </div>
</div>
```

CSS

```
body {
  margin : 0;
}

.outer-container {
  position : absolute;
  display: table;
  width: 100%; /* This could be ANY width */
  height: 100%; /* This could be ANY height */
  background: #ccc;
}

.inner-container {
  display: table-cell;
  vertical-align: middle;
  text-align: center;
}

.centered-content {
  display: inline-block;
  text-align: left;
  background: #fff;
  padding: 20px;
  border: 1px solid #000;
}
```

See also [this Fiddle!](#)

Centering with fixed size

If the size of your content is fixed, you can use absolute positioning to 50% with `margin` that reduces half of your content's width and height:

HTML

```
<div class="center">
  Center vertically and horizontally
</div>
```

CSS

```
.center {
  position: absolute;
  background: #ccc;

  left: 50%;
  width: 150px;
  margin-left: -75px; /* width * -0.5 */

  top: 50%;
  height: 200px;
  margin-top: -100px; /* height * -0.5 */
}
```

Horizontal centering with only fixed width

You can center the element horizontally even if you don't know the height of the content:

HTML

```
<div class="center">
  Center only horizontally
</div>
```

CSS

```
.center {
  position: absolute;
  background: #ccc;

  left: 50%;
  width: 150px;
  margin-left: -75px; /* width * -0.5 */
}
```

Vertical centering with fixed height

You can center the element vertically if you know the element's height:

HTML

```
<div class="center">
  Center only vertically
</div>
```

CSS

```
.center {
  position: absolute;
  background: #ccc;

  top: 50%;
  height: 200px;
}
```

```
margin-top: -100px; /* width * -0.5 */
}
```

Using margin: 0 auto;

Objects can be centered by using `margin: 0 auto;` if they are block elements and have a defined width.

HTML

```
<div class="containerDiv">
  <div id="centeredDiv"></div>
</div>

<div class="containerDiv">
  <p id="centeredParagraph">This is a centered paragraph.</p>
</div>

<div class="containerDiv">
  
</div>
```

CSS

```
.containerDiv {
  width: 100%;
  height: 100px;
  padding-bottom: 40px;
}

#centeredDiv {
  margin: 0 auto;
  width: 200px;
  height: 100px;
  border: 1px solid #000;
}

#centeredParagraph {
  width: 200px;
  margin: 0 auto;
}

#centeredImage {
  display: block;
  width: 200px;
  margin: 0 auto;
}
```

Result:



This is a centered paragraph.



JSFiddle example: [Centering objects with margin: 0 auto;](#)

Read Centering online: <https://riptutorial.com/css/topic/299/centering>

Chapter 12: Clipping and Masking

Syntax

- **Clipping**
 - clip-path: <clip-source> | [<basic-shape> || <clip-geometry-box>] | none
- **Masking**
 - mask-image: [none | <mask-reference>]#
 - mask-mode: [<mask-mode>]#
 - mask-repeat: [<repeat-style>]#
 - mask-position: [<position>]#
 - mask-clip: [<geometry-box> | no-clip]#
 - mask-origin: [<geometry-box>]#
 - mask-size: [<bg-size>]#
 - mask-composite: [<compositing-operator>]#
 - mask: [<mask-reference> <masking-mode>? || <position> [/ <bg-size>]? || <repeat-style> || <geometry-box> || [<geometry-box> | no-clip] || <compositing-operator>]#

Parameters

Parameter	Details
clip-source	A URL which can point to an inline SVG element (or) an SVG element in an external file that contains the clip path's definition.
basic-shape	Refers to one among <code>inset()</code> , <code>circle()</code> , <code>ellipse()</code> or <code>polygon()</code> . Using one of these functions the clipping path is defined. These shape functions work exactly the same way as they do in Shapes for Floats
clip-geometry-box	This can have one among <code>content-box</code> , <code>padding-box</code> , <code>border-box</code> , <code>margin-box</code> , <code>fill-box</code> , <code>stroke-box</code> , <code>view-box</code> as values. When this is provided without any value for <basic-shape>, the edges of the corresponding box is used as the path for clipping. When used with a <basic-shape>, this acts as the reference box for the shape.
mask-reference	This can be <code>none</code> or an image or a reference URL to a mask image source.
repeat-style	This specifies how the mask should be repeated or tiled in the X and Y axes. The supported values are <code>repeat-x</code> , <code>repeat-y</code> , <code>repeat</code> , <code>space</code> , <code>round</code> , <code>no-repeat</code> .
mask-mode	Can be <code>alpha</code> or <code>luminance</code> or <code>auto</code> and indicates whether the mask should be treated as a alpha mask or a luminance mask. If no value is provided and the mask-reference is a direct image then it would be considered as alpha mask (or) if the mask-reference is a URL then it would be considered as luminance

Parameter	Details
	mask.
position	This specifies the position of each mask layer and is similar in behavior to the <code>background-position</code> property. The value can be provided in 1 value syntax (like <code>top, 10%</code>) or in 2 value syntax (like <code>top right, 50% 50%</code>).
geometry-box	This specifies the box to which the mask should be clipped (<i>mask painting area</i>) or the box which should be used as reference for the mask's origin (<i>mask positioning area</i>) depending on the property. The list of possible values are <code>content-box</code> , <code>padding-box</code> , <code>border-box</code> , <code>margin-box</code> , <code>fill-box</code> , <code>stroke-box</code> , <code>view-box</code> . Detailed explanation of how each of those values work is available in the W3C Spec .
bg-size	This represents the size of each mask-image layer and has the same syntax as <code>background-size</code> . The value can be length or percentage or auto or cover or contain. Length, percentage and auto can either be provided as a single value or as one for each axis.
compositing-operator	This can be any one among <code>add</code> , <code>subtract</code> , <code>exclude</code> , <code>multiply</code> per layer and defines the type of compositing operation that should be used for this layer with those below it. Detailed explanation about each value is available in the W3C Specs .

Remarks

CSS Clipping and Masking are very new concepts and so the browser support for these properties are pretty low.

Masks:

As at the time of writing (Jul '16), Chrome, Safari and Opera support these properties with the `-webkit-` prefix.

Firefox doesn't require prefixes but it supports masks only when used with `SVG mask` elements. For inline `SVG mask` elements, the syntax is `mask: url(#msk)` whereas for using `mask` elements in an external SVG file the syntax is `mask: url('yourfilepath/yourfilename.svg#msk')`. `#msk` in both cases refers to the `id` of the `mask` element that is being referred to. As indicated in [this answer](#), at present Firefox doesn't support any parameter other than `mask-reference` in the `mask` property.

Internet Explorer (and Edge) does not offer any support for this property as yet.

The `mask-mode` property is currently not supported by any browser **with or without** prefixes.

Clip-path:

As at the time writing (Jul '16) Chrome, Safari and Opera supports `clip-path` when the path is created using basic shapes (like `circle`, `polygon`) or the `url(#clipper)` syntax with inline SVG. They don't support clipping based on shapes that are part of external SVG files. Also, they require the `-webkit` prefix to be present.

Firefox supports only the `url()` syntax for `clip-path` whereas Internet Explorer (and Edge) offer no support.

Examples

Clipping (Polygon)

CSS:

```
div{
  width:200px;
  height:200px;
  background:teal;
  clip-path: polygon(0 0, 0 100%, 100% 50%); /* refer remarks before usage */
}
```

HTML:

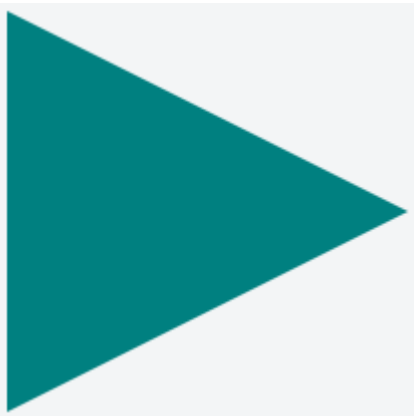
```
<div></div>
```

In the above example, a **polygonal** clipping path is used to clip the square (200 x 200) element into a triangle shape. The output shape is a triangle because the path starts at (that is, first coordinates are at) `0 0` - which is the top-left corner of the box, then goes to `0 100%` - which is bottom-left corner of the box and then finally to `100% 50%` which is nothing but the right-middle point of the box. These paths are self closing (that is, the starting point will be the ending point) and so the final shape is that of a triangle.

This can also be used on an element with an image or a gradient as background.

[View Example](#)

Output:



Clipping (Circle)

CSS:

```
div{  
  width: 200px;  
  height: 200px;  
  background: teal;  
  clip-path: circle(30% at 50% 50%); /* refer remarks before usage */  
}
```

HTML

```
<div></div>
```

This example shows how to clip a div to a circle. The element is clipped into a circle whose radius is 30% based on the dimensions of the reference box with its center point at the center of the reference box. Here since no `<clip-geometry-box>` (in other words, reference box) is provided, the `border-box` of the element will be used as the reference box.

The circle shape needs to have a radius and a center with (x, y) coordinates:

```
circle(radius at x y)
```

[View Example](#)

Output:

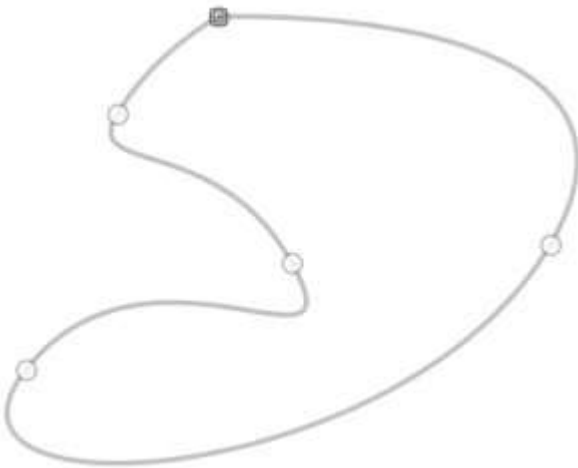


Clipping and Masking: Overview and Difference

With **Clipping** and **Masking** you can make some specified parts of elements transparent or opaque. Both can be applied to any HTML element.

Clipping

Clips are vector paths. Outside of this path the element will be transparent, inside it's opaque. Therefore you can define a `clip-path` property on elements. Every graphical element that also exists in SVG you can use here as a function to define the path. Examples are `circle()`, `polygon()` or `ellipse()`.



Example

```
clip-path: circle(100px at center);
```

The element will be only visible inside of this circle, which is positioned at the center of the element and has a radius of 100px.

Masking

Masks are similar to Clips, but instead of defining a path you define a mask what layers over the

element. You can imagine this mask as an image what consist of mainly two colors: black and white.

Luminance Mask: Black means the region is opaque, and white that it's transparent, but there is also a grey area which is semi-transparent, so you are able to make smooth transitions.

Alpha Mask: Only on the transparent areas of the mask the element will be opaque.



This image for example can be used as a luminance mask to make for an element a very smooth transition from right to left and from opaque to transparent.

The `mask` property let you specify the the mask type and an image to be used as layer.

Example

```
mask: url(masks.svg#rectangle) luminance;
```

An element called `rectangle` defined in `masks.svg` will be used as an **luminance mask** on the element.

Simple mask that fades an image from solid to transparent

CSS

```
div {  
  height: 200px;  
  width: 200px;  
  background: url(http://lorempixel.com/200/200/nature/1);  
  mask-image: linear-gradient(to right, white, transparent);  
}
```

HTML

```
<div></div>
```

In the above example there is an element with an image as its background. The mask that is

applied on the image (using CSS) makes it look as though it is fading out from left to right.

The masking is achieved by using a `linear-gradient` that goes from white (on the left) to transparent (on the right) as the mask. As it is an alpha mask, image becomes transparent where the mask is transparent.

Output without the mask:



Output with the mask:



Note: As mentioned in remarks, the above example would work in Chrome, Safari and Opera only when used with the `-webkit` prefix. This example (with a `linear-gradient` as mask image) is not yet supported in Firefox.

Using masks to cut a hole in the middle of an image

CSS

```
div {  
  width: 200px;  
  height: 200px;  
  background: url(http://lorempixel.com/200/200/abstract/6);  
  mask-image: radial-gradient(circle farthest-side at center, transparent 49%, white 50%); /*  
  check remarks before using */  
}
```


HTML

In the above example, a transparent circle is created at the center using `radial-gradient` and this is then used as a mask to produce the effect of a circle being cut out from the center of an image.

Image without mask:



Image with mask:



Using masks to create images with irregular shapes

CSS

```
div { /* check remarks before usage */
  height: 200px;
  width: 400px;
  background-image: url(http://lorempixel.com/400/200/nature/4);
  mask-image: linear-gradient(to top right, transparent 49.5%, white 50.5%), linear-
gradient(to top left, transparent 49.5%, white 50.5%), linear-gradient(white, white);
  mask-size: 75% 25%, 25% 25%, 100% 75%;
  mask-position: bottom left, bottom right, top left;
  mask-repeat: no-repeat;
}
```

HTML

```
<div></div>
```

In the above example, three `linear-gradient` images (which when placed in their appropriate positions would cover 100% x 100% of the container's size) are used as masks to produce a transparent triangular shaped cut at the bottom of the image.

Image without the mask:



Image with the mask:



Read Clipping and Masking online: <https://riptutorial.com/css/topic/3721/clipping-and-masking>

Chapter 13: Colors

Syntax

- color: #rgb
- color: #rrggbb
- color: rgb[a](<red>, <green>, <blue>[, <alpha>])
- color: hsl[a](<hue>, <saturation%>, <lightness%>[, <alpha>])
- color: **colorkeyword** /* green, blue, yellow, orange, red, ..etc */

Examples

Color Keywords

Most browsers support using color keywords to specify a color. For example, to set the `color` of an element to blue, use the `blue` keyword:

```
.some-class {  
    color: blue;  
}
```

CSS keywords are not case sensitive—`blue`, `Blue` and `BLUE` will all result in `#0000FF`.

Color Keywords

Color name	Hex value	RGB values	Color
AliceBlue	#F0F8FF	rgb(240,248,255)	
AntiqueWhite	#FAEBD7	rgb(250,235,215)	
Aqua	#00FFFF	rgb(0,255,255)	
Aquamarine	#7FFFD4	rgb(127,255,212)	
Azure	#F0FFFF	rgb(240,255,255)	
Beige	#F5F5DC	rgb(245,245,220)	
Bisque	#FFE4C4	rgb(255,228,196)	
Black	#000000	rgb(0,0,0)	

Color name	Hex value	RGB values	Color
BlanchedAlmond	#FFEBCD	rgb(255,235,205)	
Blue	#0000FF	rgb(0,0,255)	
BlueViolet	#8A2BE2	rgb(138,43,226)	
Brown	#A52A2A	rgb(165,42,42)	
BurlyWood	#DEB887	rgb(222,184,135)	
CadetBlue	#5F9EA0	rgb(95,158,160)	
Chartreuse	#7FFF00	rgb(127,255,0)	
Chocolate	#D2691E	rgb(210,105,30)	
Coral	#FF7F50	rgb(255,127,80)	
CornflowerBlue	#6495ED	rgb(100,149,237)	
Cornsilk	#FFF8DC	rgb(255,248,220)	
Crimson	#DC143C	rgb(220,20,60)	
Cyan	#00FFFF	rgb(0,255,255)	
DarkBlue	#00008B	rgb(0,0,139)	
DarkCyan	#008B8B	rgb(0,139,139)	
DarkGoldenRod	#B8860B	rgb(184,134,11)	
DarkGray	#A9A9A9	rgb(169,169,169)	
DarkGrey	#A9A9A9	rgb(169,169,169)	
DarkGreen	#006400	rgb(0,100,0)	
DarkKhaki	#BDB76B	rgb(189,183,107)	
DarkMagenta	#8B008B	rgb(139,0,139)	

Color name	Hex value	RGB values	Color
DarkOliveGreen	#556B2F	rgb(85,107,47)	
DarkOrange	#FF8C00	rgb(255,140,0)	
DarkOrchid	#9932CC	rgb(153,50,204)	
DarkRed	#8B0000	rgb(139,0,0)	
DarkSalmon	#E9967A	rgb(233,150,122)	
DarkSeaGreen	#8FBC8F	rgb(143,188,143)	
DarkSlateBlue	#483D8B	rgb(72,61,139)	
DarkSlateGray	#2F4F4F	rgb(47,79,79)	
DarkSlateGrey	#2F4F4F	rgb(47,79,79)	
DarkTurquoise	#00CED1	rgb(0,206,209)	
DarkViolet	#9400D3	rgb(148,0,211)	
DeepPink	#FF1493	rgb(255,20,147)	
DeepSkyBlue	#00BFFF	rgb(0,191,255)	
DimGray	#696969	rgb(105,105,105)	
DimGrey	#696969	rgb(105,105,105)	
DodgerBlue	#1E90FF	rgb(30,144,255)	
FireBrick	#B22222	rgb(178,34,34)	
FloralWhite	#FFFAF0	rgb(255,250,240)	
ForestGreen	#228B22	rgb(34,139,34)	
Fuchsia	#FF00FF	rgb(255,0,255)	
Gainsboro	#DCDCDC	rgb(220,220,220)	

Color name	Hex value	RGB values	Color
GhostWhite	#F8F8FF	rgb(248,248,255)	
Gold	#FFD700	rgb(255,215,0)	
GoldenRod	#DAA520	rgb(218,165,32)	
Gray	#808080	rgb(128,128,128)	
Grey	#808080	rgb(128,128,128)	
Green	#008000	rgb(0,128,0)	
GreenYellow	#ADFF2F	rgb(173,255,47)	
HoneyDew	#F0FFF0	rgb(240,255,240)	
HotPink	#FF69B4	rgb(255,105,180)	
IndianRed	#CD5C5C	rgb(205,92,92)	
Indigo	#4B0082	rgb(75,0,130)	
Ivory	#FFFFFF0	rgb(255,255,240)	
Khaki	#F0E68C	rgb(240,230,140)	
Lavender	#E6E6FA	rgb(230,230,250)	
LavenderBlush	#FFF0F5	rgb(255,240,245)	
LawnGreen	#7CFC00	rgb(124,252,0)	
LemonChiffon	#FFFACD	rgb(255,250,205)	
LightBlue	#ADD8E6	rgb(173,216,230)	
LightCoral	#F08080	rgb(240,128,128)	
LightCyan	#E0FFFF	rgb(224,255,255)	
LightGoldenRodYellow	#FAFAD2	rgb(250,250,210)	

Color name	Hex value	RGB values	Color
LightGray	#D3D3D3	rgb(211,211,211)	
LightGrey	#D3D3D3	rgb(211,211,211)	
LightGreen	#90EE90	rgb(144,238,144)	
LightPink	#FFB6C1	rgb(255,182,193)	
LightSalmon	#FFA07A	rgb(255,160,122)	
LightSeaGreen	#20B2AA	rgb(32,178,170)	
LightSkyBlue	#87CEFA	rgb(135,206,250)	
LightSlateGray	#778899	rgb(119,136,153)	
LightSlateGrey	#778899	rgb(119,136,153)	
LightSteelBlue	#B0C4DE	rgb(176,196,222)	
LightYellow	#FFFFE0	rgb(255,255,224)	
Lime	#00FF00	rgb(0,255,0)	
LimeGreen	#32CD32	rgb(50,205,50)	
Linen	#FAF0E6	rgb(250,240,230)	
Magenta	#FF00FF	rgb(255,0,255)	
Maroon	#800000	rgb(128,0,0)	
MediumAquaMarine	#66CDAA	rgb(102,205,170)	
MediumBlue	#0000CD	rgb(0,0,205)	
MediumOrchid	#BA55D3	rgb(186,85,211)	
MediumPurple	#9370DB	rgb(147,112,219)	
MediumSeaGreen	#3CB371	rgb(60,179,113)	

Color name	Hex value	RGB values	Color
MediumSlateBlue	#7B68EE	rgb(123,104,238)	
MediumSpringGreen	#00FA9A	rgb(0,250,154)	
MediumTurquoise	#48D1CC	rgb(72,209,204)	
MediumVioletRed	#C71585	rgb(199,21,133)	
MidnightBlue	#191970	rgb(25,25,112)	
MintCream	#F5FFFA	rgb(245,255,250)	
MistyRose	#FFE4E1	rgb(255,228,225)	
Moccasin	#FFE4B5	rgb(255,228,181)	
NavajoWhite	#FFDEAD	rgb(255,222,173)	
Navy	#000080	rgb(0,0,128)	
OldLace	#FDF5E6	rgb(253,245,230)	
Olive	#808000	rgb(128,128,0)	
OliveDrab	#6B8E23	rgb(107,142,35)	
Orange	#FFA500	rgb(255,165,0)	
OrangeRed	#FF4500	rgb(255,69,0)	
Orchid	#DA70D6	rgb(218,112,214)	
PaleGoldenRod	#EEE8AA	rgb(238,232,170)	
PaleGreen	#98FB98	rgb(152,251,152)	
PaleTurquoise	#AFEEEE	rgb(175,238,238)	
PaleVioletRed	#DB7093	rgb(219,112,147)	
PapayaWhip	#FFEFD5	rgb(255,239,213)	

Color name	Hex value	RGB values	Color
PeachPuff	#FFDAB9	rgb(255,218,185)	
Peru	#CD853F	rgb(205,133,63)	
Pink	#FFC0CB	rgb(255,192,203)	
Plum	#DDA0DD	rgb(221,160,221)	
PowderBlue	#B0E0E6	rgb(176,224,230)	
Purple	#800080	rgb(128,0,128)	
RebeccaPurple	#663399	rgb(102,51,153)	
Red	#FF0000	rgb(255,0,0)	
RosyBrown	#BC8F8F	rgb(188,143,143)	
RoyalBlue	#4169E1	rgb(65,105,225)	
SaddleBrown	#8B4513	rgb(139,69,19)	
Salmon	#FA8072	rgb(250,128,114)	
SandyBrown	#F4A460	rgb(244,164,96)	
SeaGreen	#2E8B57	rgb(46,139,87)	
SeaShell	#FFF5EE	rgb(255,245,238)	
Sienna	#A0522D	rgb(160,82,45)	
Silver	#C0C0C0	rgb(192,192,192)	
SkyBlue	#87CEEB	rgb(135,206,235)	
SlateBlue	#6A5ACD	rgb(106,90,205)	
SlateGray	#708090	rgb(112,128,144)	
SlateGrey	#708090	rgb(112,128,144)	

Color name	Hex value	RGB values	Color
Snow	#FFFAFA	rgb(255,250,250)	
SpringGreen	#00FF7F	rgb(0,255,127)	
SteelBlue	#4682B4	rgb(70,130,180)	
Tan	#D2B48C	rgb(210,180,140)	
Teal	#008080	rgb(0,128,128)	
Thistle	#D8BFD8	rgb(216,191,216)	
Tomato	#FF6347	rgb(255,99,71)	
Turquoise	#40E0D0	rgb(64,224,208)	
Violet	#EE82EE	rgb(238,130,238)	
Wheat	#F5DEB3	rgb(245,222,179)	
White	#FFFFFF	rgb(255,255,255)	
WhiteSmoke	#F5F5F5	rgb(245,245,245)	
Yellow	#FFFF00	rgb(255,255,0)	
YellowGreen	#9ACD32	rgb(154,205,50)	

In addition to the named colors, there is also the keyword `transparent`, which represents a fully-transparent black: `rgba(0,0,0,0)`

Hexadecimal Value

Background

CSS colors may also be represented as a hex triplet, where the members represent the red, green and blue components of a color. Each of these values represents a number in the range of `00` to `FF`, or `0` to `255` in decimal notation. Uppercase and/or lowercase Hexidecimal values may be used (i.e. `#3fc` = `#3FC` = `#33ffcc`). The browser interprets `#369` as `#336699`. If that is not what you intended but rather wanted `#306090`, you need to specify that explicitly.

The total number of colors that can be represented with hex notation is 256^3 or 16,777,216.

Syntax

```
color: #rrggbb;  
color: #rgb
```

Value	Description
rr	00 - FF for the amount of red
gg	00 - FF for the amount of green
bb	00 - FF for the amount of blue

```
.some-class {  
  /* This is equivalent to using the color keyword 'blue' */  
  color: #0000FF;  
}  
  
.also-blue {  
  /* If you want to specify each range value with a single number, you can!  
   This is equivalent to '#0000FF' (and 'blue') */  
  color: #00F;  
}
```

[Hexadecimal notation](#) is used to specify color values in the RGB color format, per the [W3C's 'Numerical color values'](#).

There are a lot of tools available on the Internet for looking up hexadecimal (or simply hex) color values.

Search for "**hex color palette**" or "**hex color picker**" with your favorite web browser to find a bunch of options!

Hex values always start with a pound sign (#), are up to six "digits" long, and are case-insensitive: that is, they don't care about capitalization. #FFC125 and #ffc125 are the same color.

rgb() Notation

RGB is an additive color model which represents colors as mixtures of red, green, and blue light. In essence, the RGB representation is the decimal equivalent of the Hexadecimal Notation. In Hexadecimal each number ranges from 00-FF which is equivalent to 0-255 in decimal and 0%-100% in percentages.

```
.some-class {  
  /* Scalar RGB, equivalent to 'blue'*/  
  color: rgb(0, 0, 255);  
}
```

```
.also-blue {  
    /* Percentile RGB values*/  
    color: rgb(0%, 0%, 100%);  
}
```

Syntax

```
rgb(<red>, <green>, <blue>)
```

Value	Description
<red>	an integer from 0 - 255 or percentage from 0 - 100%
<green>	an integer from 0 - 255 or percentage from 0 - 100%
<blue>	an integer from 0 - 255 or percentage from 0 - 100%

hsl() Notation

HSL stands for **hue** ("which color"), **saturation** ("how much color") and **lightness** ("how much white").

Hue is represented as an angle from 0° to 360° (without units), while saturation and lightness are represented as percentages.

```
p {  
    color: hsl(240, 100%, 50%); /* Blue */  
}
```

Syntax

```
color: hsl(<hue>, <saturation>%, <lightness>%);
```

Value	Description
<hue>	specified in degrees around the color wheel (without units), where 0° is red, 60° is yellow, 120° is green, 180° is cyan, 240° is blue, 300° is magenta, and 360° is red
<saturation>	specified in percentage where 0% is fully desaturated (grayscale) and 100% is fully saturated (vividly colored)
<lightness>	specified in percentage where 0% is fully black and 100% is fully white

Notes

- A saturation of 0% always produces a grayscale color; changing the hue has no effect.
- A lightness of 0% always produces black, and 100% always produces white; changing the hue or saturation has no effect.

currentColor

`currentColor` returns the computed color value of the current element.

Use in same element

Here `currentColor` evaluates to red since the `color` property is set to `red`:

```
div {  
  color: red;  
  border: 5px solid currentColor;  
  box-shadow: 0 0 5px currentColor;  
}
```

In this case, specifying `currentColor` for the border is most likely redundant because omitting it should produce identical results. Only use `currentColor` inside the border property within the same element if it would be overwritten otherwise due to a [more specific](#) selector.

Since it's the computed color, the border will be green in the following example due to the second rule overriding the first:

```
div {  
  color: blue;  
  border: 3px solid currentColor;  
  color: green;  
}
```

Inherited from parent element

The parent's color is inherited, here `currentColor` evaluates to 'blue', making the child element's border-color blue.

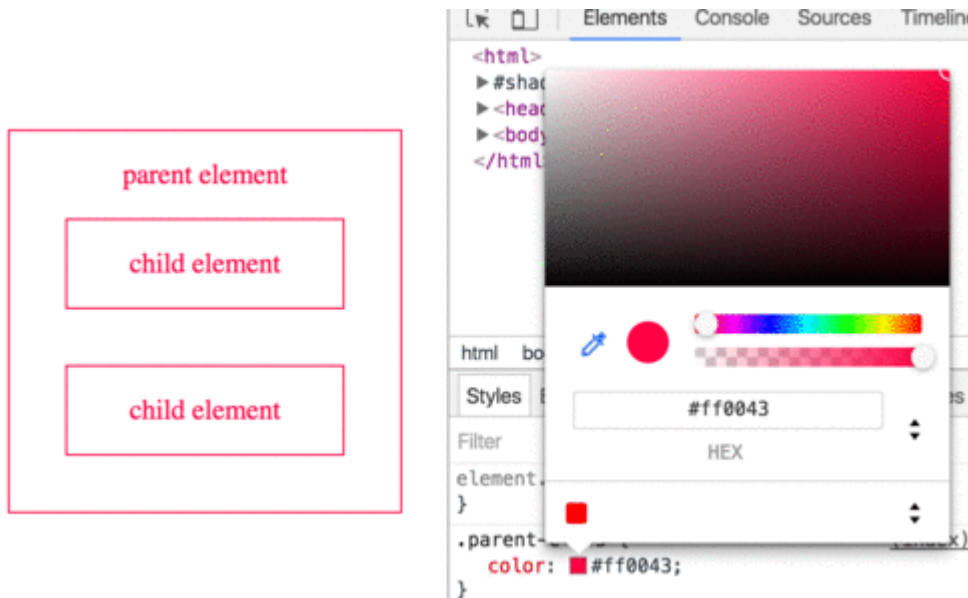
```
.parent-class {  
  color: blue;  
}  
  
.parent-class .child-class {  
  border-color: currentColor;  
}
```

`currentColor` can also be used by other rules which normally would not inherit from the color

property, such as background-color. The example below shows the children using the color set in the parent as its background:

```
.parent-class {  
  color: blue;  
}  
  
.parent-class .child-class {  
  background-color: currentColor;  
}
```

Possible Result:



rgba() Notation

Similar to [rgb\(\) notation](#), but with an additional alpha (opacity) value.

```
.red {  
  /* Opaque red */  
  color: rgba(255, 0, 0, 1);  
}  
  
.red-50p {  
  /* Half-translucent red. */  
  color: rgba(255, 0, 0, .5);  
}
```

Syntax

```
rgba(<red>, <green>, <blue>, <alpha>);
```

Value	Description
<red>	an integer from 0 - 255 or percentage from 0 - 100%

Value	Description
<green>	an integer from 0 - 255 or percentage from 0 - 100%
<blue>	an integer from 0 - 255 or percentage from 0 - 100%
<alpha>	a number from 0 - 1, where 0.0 is fully transparent and 1.0 is fully opaque

hsla() Notation

Similar to [hsl\(\) notation](#), but with an added alpha (opacity) value.

```
hsla(240, 100%, 50%, 0)    /* transparent */
hsla(240, 100%, 50%, 0.5) /* half-translucent blue */
hsla(240, 100%, 50%, 1)   /* fully opaque blue */
```

Syntax

```
hsla(<hue>, <saturation>%, <lightness>%, <alpha>);
```

Value	Description
<hue>	specified in degrees around the color wheel (without units), where 0° is red, 60° is yellow, 120° is green, 180° is cyan, 240° is blue, 300° is magenta, and 360° is red
<saturation>	percentage where 0% is fully desaturated (grayscale) and 100% is fully saturated (vividly colored)
<lightness>	percentage where 0% is fully black and 100% is fully white
<alpha>	a number from 0 - 1 where 0 is fully transparent and 1 is fully opaque

Read Colors online: <https://riptutorial.com/css/topic/644/colors>

Chapter 14: Columns

Syntax

- `column-count`: `auto`|`number`|`inherit`|`initial`|`unset`;
- `column-width`: `auto`|`length`;
- `column`: [`column-width`][`column-count`];
- `column-span`: `none`|`all`|`inherit`|`initial`|`unset`;
- `column-gap`: `normal`|`length`|`inherit`|`initial`|`unset`;
- `column-fill`: `auto`|`balance`|`inherit`|`initial`|`unset`;
- `column-rule-color`: `color`|`inherit`|`initial`|`unset`;
- `column-rule-style`:
`none`|`hidden`|`dotted`|`dashed`|`solid`|`double`|`groove`|`ridge`|`inset`|`outset`|`inherit`|`initial`|`unset`;
- `column-rule-width`: `thin`|`medium`|`thick`|`length`|`inherit`|`initial`|`unset`;
- `column-rule`: [`column-rule-width`][`column-rule-style`][`column-rule-color`];
- `break-after`: `auto`|`always`|`left`|`right`|`recto`|`verso`|`page`|`column`|`region`|`avoid`|`avoid-page`|`avoid-column`|`avoid-region`;
- `break-before`: `auto`|`always`|`left`|`right`|`recto`|`verso`|`page`|`column`|`region`|`avoid`|`avoid-page`|`avoid-column`|`avoid-region`;
- `break-inside`: `auto`|`avoid`|`avoid-page`|`avoid-column`|`avoid-region`;

Examples

Simple Example (column-count)

The CSS multi-column layout makes it easy to create multiple columns of text.

Code

```
<div id="multi-columns">Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do  
eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis  
nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure  
dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.  
Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim  
id est laborum</div>
```

```
.multi-columns {  
  -moz-column-count: 2;  
  -webkit-column-count: 2;  
  column-count: 2;  
}
```

Result

Stack Overflow is a privately held website, the flagship site of the Stack Exchange Network,[4][5][6] created in 2008 by Jeff Atwood and Joel Spolsky. [7][8] It was created to be a more open alternative to earlier Q&A sites such as Experts-Exchange. The name for the website was chosen by voting in April 2008 by readers of Coding Horror, Atwood's popular programming blog.

The website serves as a platform for users to ask and answer questions, and, through membership and active participation, to vote questions and

answers and answers. It is a wiki or Discreet Question and Answer Overflow. It has "badges" awarded to users for receiving answers. Badges are given to a user for receiving a certain number of badges for a certain number of [14] which is a form of gamification. It is a forum. It is licensed under the Creative Commons Attribution-ShareAlike license.

Column Width

The `column-width` property sets the minimum column width. If `column-count` is not defined the browser will make as many columns as fit in the available width.

Code:

```
<div id="multi-columns">
  Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt
  ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco
  laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in
  voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat
  non proident, sunt in culpa qui officia deserunt mollit anim id est laborum
</div>
```

```
.multi-columns {
  -moz-column-width: 100px;
  -webkit-column-width: 100px;
  column-width: 100px;
}
```

Result

Stack Overflow is a privately held website, the flagship site of the Stack Exchange Network,[4][5][6] created in 2008 by Jeff Atwood and Joel Spolsky. [7][8] It was created to be a more open alternative to earlier Q&A sites such as Experts-	Exchange. The name for the website was chosen by voting in April 2008 by readers of Coding Horror, Atwood's popular programming blog. The website serves as a platform for users to ask and answer	questions, and, through membership and active participation, to vote questions and answers up or down and edit questions and answers in a fashion similar to a wiki or Digg.[13] Users of Stack Overflow can earn reputation	points and "badges"; for example, a person is awarded 10 reputation points for receiving an "up" vote on an answer given to a question, and can receive badges for their valued contributions, [14] which represents a kind of	gamification of the traditional Q&A site or forum. All user-generated content is licensed under a Creative Commons Attribute-ShareAlike license.
--	--	--	--	--

Read Columns online: <https://riptutorial.com/css/topic/3042/columns>

Chapter 15: Comments

Syntax

- `/* Comment */`

Remarks

- Comments in CSS always start with `/*` and end with `*/`
- Comments cannot be nested

Examples

Single Line

```
/* This is a CSS comment */
div {
    color: red; /* This is a CSS comment */
}
```

Multiple Line

```
/*
    This
    is
    a
    CSS
    comment
*/
div {
    color: red;
}
```

Read Comments online: <https://riptutorial.com/css/topic/1625/comments>

Chapter 16: Counters

Syntax

- counter-set: [<counter-name> <integer>?]+ | none
- counter-reset: [<counter-name> <integer>?]+ | none
- counter-increment: [<counter-name> <integer>?]+ | none
- counter(<counter-name> [, <counter-style>]?)
- counters(<counter-name>, <connector-string> [, <counter-style>]?)

Parameters

Parameter	Details
counter-name	This is the name of the counter that needs to be created or incremented or printed. It can be any custom name as the developer wishes.
integer	This integer is an optional value that when provided next to the counter name will represent the initial value of the counter (in <code>counter-set</code> , <code>counter-reset</code> properties) or the value by which the counter should be incremented (in <code>counter-increment</code>).
none	This is the initial value for all 3 <code>counter-*</code> properties. When this value is used for <code>counter-increment</code> , the value of none of the counters are affected. When this is used for the other two, no counter is created.
counter-style	This specifies the style in which the counter value needs to be displayed. It supports all values supported by the <code>list-style-type</code> property. If <code>none</code> is used then the counter value is not printed at all.
connector-string	This represents the string that must be placed between the values of two different counter levels (like the "." in "2.1.1").

Remarks

Counters are not a new topic in CSS. It was a part of the CSS Level 2 Specifications (Revision 1 to be precise) itself and hence has very high browser support.

All browsers except IE6 and IE7 have support for CSS Counters.

Examples

Applying roman numerals styling to the counter output

CSS

```
body {
  counter-reset: item-counter;
}

.item {
  counter-increment: item-counter;
}

.item:before {
  content: counter(item-counter, upper-roman) ". "; /* by specifying the upper-roman as style
the output would be in roman numbers */
}
```

HTML

```
<div class='item'>Item No: 1</div>
<div class='item'>Item No: 2</div>
<div class='item'>Item No: 3</div>
```

In the above example, the counter's output would be displayed as I, II, III (roman numbers) instead of the usual 1, 2, 3 as the developer has explicitly specified the counter's style.

Number each item using CSS Counter

CSS

```
body {
  counter-reset: item-counter; /* create the counter */
}

.item {
  counter-increment: item-counter; /* increment the counter every time an element with class
"item" is encountered */
}

.item-header:before {
  content: counter(item-counter) ". "; /* print the value of the counter before the header and
append a "." to it */
}

/* just for demo */

.item {
  border: 1px solid;
  height: 100px;
  margin-bottom: 10px;
}

.item-header {
```

```
border-bottom: 1px solid;
height: 40px;
line-height: 40px;
padding: 5px;
}
.item-content {
padding: 8px;
}
```

HTML

```
<div class='item'>
  <div class='item-header'>Item 1 Header</div>
  <div class='item-content'>Lorem Ipsum Dolor Sit Amet....</div>
</div>
<div class='item'>
  <div class='item-header'>Item 2 Header</div>
  <div class='item-content'>Lorem Ipsum Dolor Sit Amet....</div>
</div>
<div class='item'>
  <div class='item-header'>Item 3 Header</div>
  <div class='item-content'>Lorem Ipsum Dolor Sit Amet....</div>
</div>
```

The above example numbers every "item" in the page and adds the item's number before its header (using `content` property of `.item-header` element's `:before` pseudo). A live demo of this code is available [here](#).

Implementing multi-level numbering using CSS counters

CSS

```
ul {
list-style: none;
counter-reset: list-item-number; /* self nesting counter as name is same for all levels */
}
li {
counter-increment: list-item-number;
}
li:before {
content: counters(list-item-number, ".") " "; /* usage of counters() function means value of
counters at all higher levels are combined before printing */
}
```

HTML

```
<ul>
  <li>Level 1
    <ul>
```

```

    <li>Level 1.1
      <ul>
        <li>Level 1.1.1</li>
      </ul>
    </li>
  </ul>
</li>
<li>Level 2
  <ul>
    <li>Level 2.1
      <ul>
        <li>Level 2.1.1</li>
        <li>Level 2.1.2</li>
      </ul>
    </li>
  </ul>
</li>
<li>Level 3</li>
</ul>

```

The above is an example of multi-level numbering using CSS counters. It makes use of the **self-nesting** concept of counters. Self nesting is a concept where if an element already has a counter with the given name but is having to create another then it creates it as a child of the existing counter. Here, the second level `ul` already inherits the `list-item-number` counter from its parent but then has to create its own `list-item-number` (for its children `li`) and so creates `list-item-number[1]` (counter for second level) and nests it under `list-item-number[0]` (counter for first level). Thus it achieves the multi-level numbering.

The output is printed using the `counters()` function instead of the `counter()` function because the `counters()` function is designed to prefix the value of all higher level counters (parent) when printing the output.

Read Counters online: <https://riptutorial.com/css/topic/2575/counters>

Chapter 17: CSS design patterns

Introduction

These examples are for documenting CSS-specific design patterns like [BEM](#), [OOCSS](#) and [SMACSS](#).

These examples are NOT for documenting CSS frameworks like [Bootstrap](#) or [Foundation](#).

Remarks

These examples are for documenting CSS-specific methodologies / design patterns.

These methodologies include but are not exclusive to the following:

- [BEM](#)
- [OOCSS](#)
- [SMACSS](#)

These examples are NOT for documenting CSS frameworks like [Bootstrap](#) or [Foundation](#). While you may include examples of how to apply one or more CSS methodology / design pattern with a CSS framework, those examples are to focus on the methodologies / design patterns with that particular framework and on the use of the framework itself.

Examples

BEM

[BEM](#) stands for `Blocks`, `Elements` and `Modifiers`. It's a methodology initially conceived by Russian tech company [Yandex](#), but which gained quite some traction among American & Western-European web developers as well.

As the same implies, BEM methodology is all about componentization of your HTML and CSS code into three types of components:

- **Blocks:** standalone entities that are meaningful on their own

Examples are `header`, `container`, `menu`, `checkbox` & `textbox`

- **Elements:** Part of blocks that have no standalone meaning and are semantically tied to their blocks.

Examples are `menu item`, `list item`, `checkbox caption` & `header title`

- **Modifiers:** Flags on a block or element, used to change appearance or behavior

Examples are disabled, highlighted, checked, fixed, size big & color yellow

The goal of BEM is to keep optimize the readability, maintainability and flexibility of your CSS code. The way to achieve this, is to apply the following rules.

- Block styles are never dependent on other elements on a page
- Blocks should have a simple, short name and avoid _ or - characters
- When styling elements, use selectors of format `blockname__elementname`
- When styling modifiers, use selectors of format `blockname--modifiername` and `blockname__elementname--modifiername`
- Elements or blocks that have modifiers should inherit everything from the block or element it is modifying except the properties the modifier is supposed to modify

Code example

If you apply BEM to your form elements, your CSS selectors should look something like this:

```
.form { } // Block
.form--theme-xmas { } // Block + modifier
.form--simple { } // Block + modifier
.form__input { } // Block > element
.form__submit { } // Block > element
.form__submit--disabled { } // Block > element + modifier
```

The corresponding HTML should look something like this:

```
<form class="form form--theme-xmas form--simple">
  <input class="form__input" type="text" />
  <input class="form__submit form__submit--disabled" type="submit" />
</form>
```

Read CSS design patterns online: <https://riptutorial.com/css/topic/10823/css-design-patterns>

Chapter 18: CSS Image Sprites

Syntax

- **//Using background-position**
background: url("sprite-image.png");
background-position: -20px 50px;
- **//Background property shorthand**
background: url("sprite-image.png") -20px 50px;

Remarks

For some use cases, sprites are slowly falling out of favor, being replaced by icon webfonts or [SVG images](#).

Examples

A Basic Implementation

What's an image sprite?

An image sprite is a single asset located within an image sprite sheet. An image sprite sheet is an image file that contains more than one asset that can be extracted from it.

For example:



The image above is an image sprite sheet, and each one of those stars is a sprite within the sprite sheet. These sprite sheets are useful because they improve performance by reducing the number of HTTP requests a browser might have to make.

So how do you implement one? Here's some example code.

HTML

```
<div class="icon icon1"></div>  
<div class="icon icon2"></div>
```

```
<div class="icon icon3"></div>
```

CSS

```
.icon {  
    background: url("icons-sprite.png");  
    display: inline-block;  
    height: 20px;  
    width: 20px;  
}  
.icon1 {  
    background-position: 0px 0px;  
}  
.icon2 {  
    background-position: -20px 0px;  
}  
.icon3 {  
    background-position: -40px 0px;  
}
```

By using setting the sprite's width and height and by using the background-position property in CSS (with an x and y value) you can easily extract sprites from a sprite sheet using CSS.

Read CSS Image Sprites online: <https://riptutorial.com/css/topic/3690/css-image-sprites>

Chapter 19: CSS Object Model (CSSOM)

Remarks

The CSS Object Model (CSSOM) is a specification on its own.

The current draft can be found here: <https://www.w3.org/TR/cssom-1/>

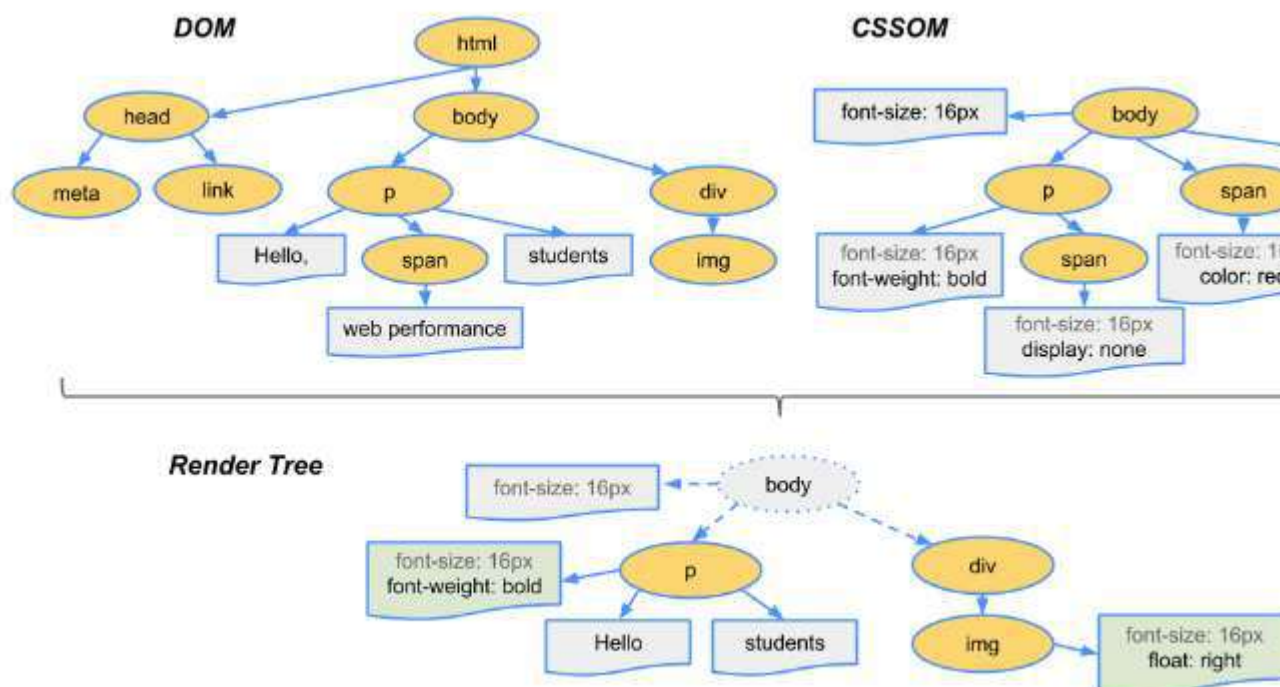
Examples

Introduction

The browser identifies tokens from stylesheet and converts them into nodes which are linked into a tree structure. The entire map of all the nodes with their associated styles of a page would be the CSS Object Model.

To display the webpage, a web browser takes following steps.

1. The web browser examines your HTML and builds the DOM (Document Object Model).
2. The web browser examines your CSS and builds the CSSOM (CSS Object Model).
3. The web browser combines the DOM and the CSSOM to create a render tree. The web browser displays your webpage.



Adding a background-image rule via the CSSOM

To add a background-image rule via the CSSOM, first get a reference to the rules of the first

stylesheet:

```
var stylesheet = document.styleSheets[0].cssRules;
```

Then, get a reference to the end of the stylesheet:

```
var end = stylesheet.length - 1;
```

Finally, insert a background-image rule for the body element at the end of the stylesheet:

```
stylesheet.insertRule("body { background-image:  
url('http://cdn.sstatic.net/Sites/stackoverflow/img/favicon.ico'); }", end);
```

Read CSS Object Model (CSSOM) online: <https://riptutorial.com/css/topic/4961/css-object-model-cssom->

Chapter 20: Cursor Styling

Syntax

- cursor: auto | default | none | context-menu | help | pointer | progress | wait | cell | crosshair | text | vertical-text | alias | copy | move | no-drop | not-allowed | e-resize | n-resize | ne-resize | nw-resize | s-resize | se-resize | sw-resize | w-resize | ew-resize | ns-resize | nesw-resize | nwse-resize | col-resize | row-resize | all-scroll | zoom-in | zoom-out | grab | grabbing;

Examples

Changing cursor type

```
cursor: value;
```

	default		n-resize		not-allowed
	crosshair		ne-resize		no-drop
	hand		e-resize		vertical-text
	pointer		se-resize		all-scroll
	Cross browser		s-resize		col-resize
	move		sw-resize		row-resize
	text		w-resize		
	wait		nw-resize		
	help		progress		

Examples:

Value	Description
none	No cursor is rendered for the element
auto	Default. The browser sets a cursor
help	The cursor indicates that help is available
wait	The cursor indicates that the program is busy
move	The cursor indicates something is to be moved
pointer	The cursor is a pointer and indicates a link

pointer-events

The pointer-events property allows for control over how HTML elements respond to mouse/touch events.

```
.disabled {  
  pointer-events: none;  
}
```

In this example,

'none' prevents all click, state and cursor options on the specified HTML element [[1]]

Other valid values for HTML elements are:

- auto;
- inherit.

1. <https://css-tricks.com/almanac/properties/p/pointer-events/>

Other resources:

- <https://developer.mozilla.org/en-US/docs/Web/CSS/pointer-events>
- <https://davidwalsh.name/pointer-events>

caret-color

The caret-color CSS property specifies the color of the caret, the visible indicator of the insertion point in an element where text and other content is inserted by the user's typing or editing.

HTML

```
<input id="example" />
```

CSS

```
#example {  
  caret-color: red;  
}
```

Resources:

- <https://developer.mozilla.org/en-US/docs/Web/CSS/caret-color>

Read Cursor Styling online: <https://riptutorial.com/css/topic/1742/cursor-styling>

Chapter 21: Custom Properties (Variables)

Introduction

CSS Variables allow authors to create reusable values which can be used throughout a CSS document.

For example, it's common in CSS to reuse a single color throughout a document. Prior to CSS Variables this would mean reusing the same color value many times throughout a document. With CSS Variables the color value can be assigned to a variable and referenced in multiple places. This makes changing values easier and is more semantic than using traditional CSS values.

Syntax

- `:root {}` /* pseudo-class that allows for more global definition of variables */
- `--variable-name: value;` /* define variable */
- `var(--variable-name, default-value)` /* use defined variable with default value fallback */

Remarks

CSS Variables are currently considered an experimental technology.

BROWSER SUPPORT / COMPATIBILITY

Firefox: Version **31+** (*Enabled by default*)

[More info from Mozilla](#)

Chrome: Version **49+** (*Enabled by default*).

"This feature can be enabled in Chrome Version 48 for testing by enabling the `experimental Web Platform` feature. Enter `chrome://flags/` in your Chrome address bar to access this setting."

IE: Not Supported.

Edge: [Under Development](#)

Safari: Version **9.1+**

Examples

Variable Color

```
:root {
```



```

--red: #b00;
--blue: #4679bd;
--grey: #ddd;
}
.Bx1 {
  color: var(--red);
  background: var(--grey);
  border: 1px solid var(--red);
}

```

Variable Dimensions

```

:root {
  --W200: 200px;
  --W10: 10px;
}
.Bx2 {
  width: var(--W200);
  height: var(--W200);
  margin: var(--W10);
}

```

Variable Cascading

CSS variables cascade in much the same way as other properties, and can be restated safely.

You can define variables multiple times and only the definition with the highest specificity will apply to the element selected.

Assuming this HTML:

```

<a class="button">Button Green</a>
<a class="button button_red">Button Red</a>
<a class="button">Button Hovered On</a>

```

We can write this CSS:

```

.button {
  --color: green;
  padding: .5rem;
  border: 1px solid var(--color);
  color: var(--color);
}

.button:hover {
  --color: blue;
}

.button_red {
  --color: red;
}

```

And get this result:



Valid/Invalids

Naming When naming CSS variables, it contains only letters and dashes just like other CSS properties (eg: line-height, -moz-box-sizing) but it should start with double dashes (--)

```
//These are Invalids variable names
--123color: blue;
--#color: red;
--bg_color: yellow
--$width: 100px;

//Valid variable names
--color: red;
--bg-color: yellow
--width: 100px;
```

CSS Variables are case sensitive.

```
/* The variable names below are all different variables */
--pcolor: ;
--Pcolor: ;
--pColor: ;
```

Empty Vs Space

```
/* Invalid */
--color;;

/* Valid */
--color: ; /* space is assigned */
```

Concatenations

```
/* Invalid - CSS doesn't support concatenation*/
.logo{
  --logo-url: 'logo';
  background: url('assets/img/' var(--logo-url) '.png');
}

/* Invalid - CSS bug */
.logo{
  --logo-url: 'assets/img/logo.png';
  background: url(var(--logo-url));
}

/* Valid */
.logo{
  --logo-url: url('assets/img/logo.png');
  background: var(--logo-url);
}
```

Careful when using Units

```
/* Invalid */
--width: 10;
width: var(--width)px;

/* Valid */
--width: 10px;
width: var(--width);

/* Valid */
--width: 10;
width: calc(1px * var(--width)); /* multiply by 1 unit to convert */
width: calc(1em * var(--width));
```

With media queries

You can re-set variables within media queries and have those new values cascade wherever they are used, something that isn't possible with pre-processor variables.

Here, a media query changes the variables used to set up a very simple grid:

HTML

```
<div></div>
<div></div>
<div></div>
<div></div>
```

CSS

```
:root{
  --width: 25%;
  --content: 'This is desktop';
}
@media only screen and (max-width: 767px){
  :root{
    --width:50%;
    --content: 'This is mobile';
  }
}
@media only screen and (max-width: 480px){
  :root{
    --width:100%;
  }
}

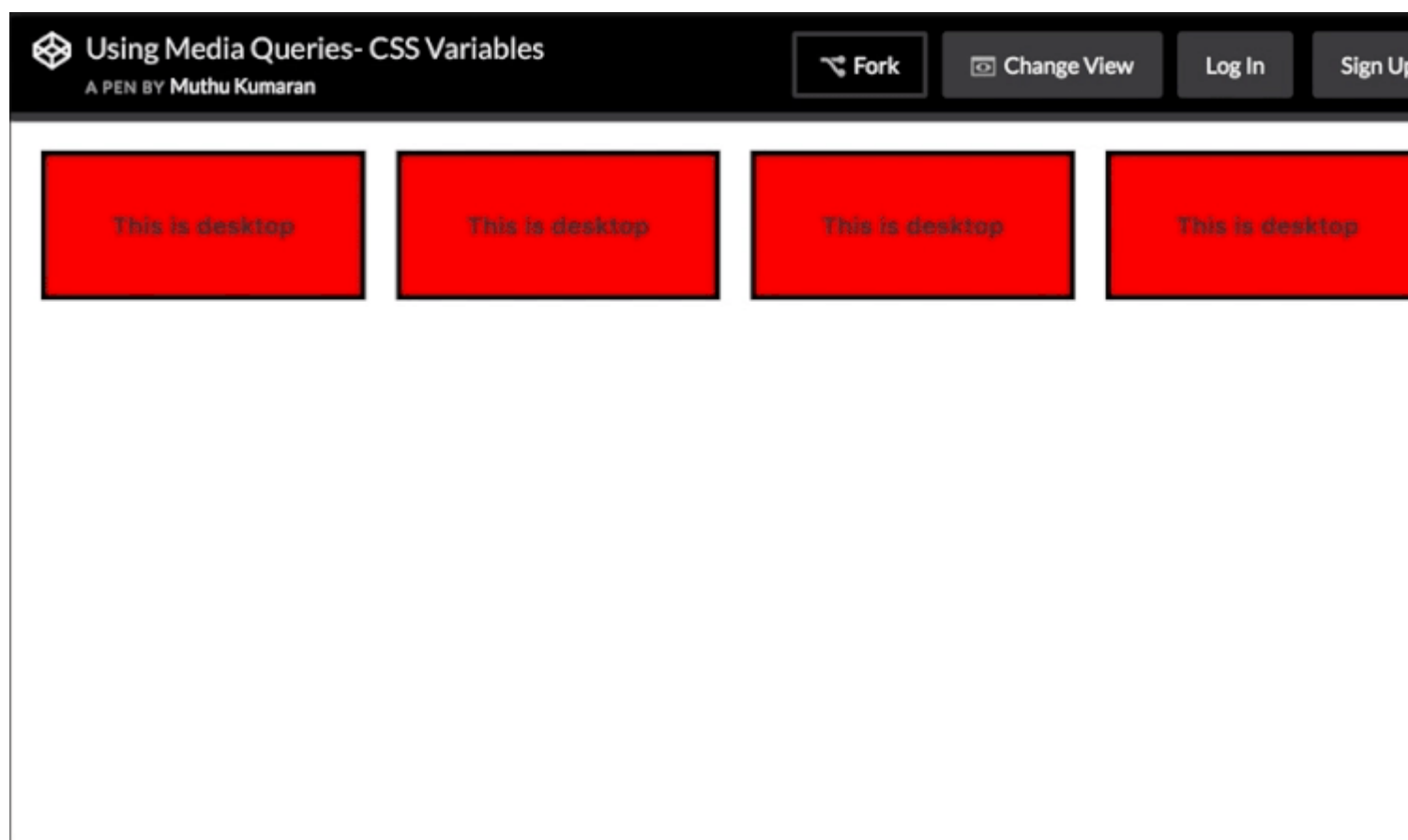
div{
  width: calc(var(--width) - 20px);
  height: 100px;
}
div:before{
  content: var(--content);
}

/* Other Styles */
```

```
body {  
  padding: 10px;  
}  
  
div{  
  display: flex;  
  align-items: center;  
  justify-content: center;  
  font-weight:bold;  
  float:left;  
  margin: 10px;  
  border: 4px solid black;  
  background: red;  
}
```

You can try resizing the window in this [CodePen Demo](#)

Here's an animated screenshot of the resizing in action:



Read Custom Properties (Variables) online: <https://riptutorial.com/css/topic/1755/custom-properties--variables->

Chapter 22: Feature Queries

Syntax

- `@supports [condition] { /* CSS rules to apply */ }`

Parameters

Parameter	Details
<code>(property: value)</code>	Evaluates true if the browser can handle the CSS rule. The parenthesis around the rule are required.
<code>and</code>	Returns true only if both the previous and next conditions are true.
<code>not</code>	Negates the next condition
<code>or</code>	Returns true if either the previous or next condition is true.
<code>(...)</code>	Groups conditions

Remarks

Feature detection using `@supports` is supported in Edge, Chrome, Firefox, Opera, and Safari 9 and up.

Examples

Basic `@supports` usage

```
@supports (display: flex) {  
  /* Flexbox is available, so use it */  
  .my-container {  
    display: flex;  
  }  
}
```

In terms of syntax, `@supports` is very similar to `@media`, but instead of detecting screen size and orientation, `@supports` will detect whether the browser can handle a given CSS rule.

Rather than doing something like `@supports (flex)`, notice that the rule is `@supports (display: flex)`.

Chaining feature detections

To detect multiple features at once, use the `and` operator.

```
@supports (transform: translateZ(1px)) and (transform-style: preserve-3d) and (perspective: 1px) {  
  /* Probably do some fancy 3d stuff here */  
}
```

There is also an `or` operator and a `not` operator:

```
@supports (display: flex) or (display: table-cell) {  
  /* Will be used if the browser supports flexbox or display: table-cell */  
}  
@supports not (-webkit-transform: translate(0, 0, 0)) {  
  /* Will *not* be used if the browser supports -webkit-transform: translate(...) */  
}
```

For the ultimate `@supports` experience, try grouping logical expressions with parenthesis:

```
@supports ((display: block) and (zoom: 1)) or ((display: flex) and (not (display: table-cell))) or (transform: translateX(1px)) {  
  /* ... */  
}
```

This will work if the browser

1. Supports `display: block` **AND** `zoom: 1`, or
2. Supports `display: flex` **AND NOT** `display: table-cell`, or
3. Supports `transform: translateX(1px)`.

Read Feature Queries online: <https://riptutorial.com/css/topic/5024/feature-queries>

Chapter 23: Filter Property

Syntax

- filter: none (default value)
- filter: initial (defaults to none);
- filter: inherit (defaults to parent value);
- filter: blur(px)
- filter: brightness(number | %)
- filter: contrast(number | %)
- filter: drop-shadow(horizontal-shadow-px vertical-shadow-px shadow-blur-px shadow-spread color)
- filter: greyscale(number | %)
- filter: hue-rotate(deg)
- filter: invert(number | %)
- filter: opacity(number | %)
- filter: saturate(number | %)
- filter: sepia(number | %)

Parameters

Value	Description
blur(x)	Blurs the image by x pixels.
brightness(x)	Brightens the image at any value above 1.0 or 100%. Below that, the image will be darkened.
contrast(x)	Provides more contrast to the image at any value above 1.0 or 100%. Below that, the image will get less saturated.
drop-shadow(h, v, x, y, z)	Gives the image a drop-shadow. h and v can have negative values. x, y, and z are optional.
greyscale(x)	Shows the image in greyscale, with a maximum value of 1.0 or 100%.
hue-rotate(x)	Applies a hue-rotation to the image.
invert(x)	Inverts the color of the image with a maximum value of 1.0 or 100%.
opacity(x)	Sets how opaque/transparent the image is with a maximum value of 1.0 or 100%.
saturate(x)	Saturates the image at any value above 1.0 or 100%. Below that, the image will start to de-saturate.

Value	Description
sepia(x)	Converts the image to sepia with a maximum value of 1.0 or 100%.

Remarks

1. Since filter is an experimental feature, you should use the -webkit prefix. It may change in syntax and behavior, but the changes are probably going to be small.
2. It might not be supported in older versions of major browsers. It might be entirely unsupported in mobile browsers.
3. Due to its relatively limited support, try to use `box-shadow` instead of `filter: drop-shadow()`. Use `opacity` instead of `filter: opacity()`.
4. It can be animated through Javascript/jQuery. For Javascript, use `object.style.WebkitFilter`.
5. Check [W3Schools](#) or [MDN](#) for more info.
6. W3Schools also has a [demo page](#) for all the different type of filter values.

Examples

Drop Shadow (use box-shadow instead if possible)

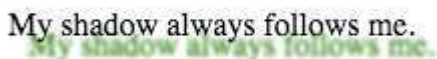
HTML

```
<p>My shadow always follows me.</p>
```

CSS

```
p {
  -webkit-filter: drop-shadow(10px 10px 1px green);
  filter: drop-shadow(10px 10px 1px green);
}
```

Result



My shadow always follows me.

Multiple Filter Values

To use multiple filters, separate each value with a space.

HTML


```
<img src='donald-duck.png' alt='Donald Duck' title='Donald Duck' />
```

CSS

```
img {  
  -webkit-filter: brightness(200%) grayscale(100%) sepia(100%) invert(100%);  
  filter: brightness(200%) grayscale(100%) sepia(100%) invert(100%);  
}
```

Result



Hue Rotate

HTML

```
<img src='donald-duck.png' alt='Donald Duck' title='Donald Duck' />
```

CSS

```
img {  
  -webkit-filter: hue-rotate(120deg);  
  filter: hue-rotate(120deg);  
}
```

Result



Invert Color

HTML

```
<div></div>
```

CSS

```
div {  
  width: 100px;  
  height: 100px;  
  background-color: white;  
  -webkit-filter: invert(100%);  
  filter: invert(100%);  
}
```

Result



Turns from white to black.

Blur

HTML

```
<img src='donald-duck.png' alt='Donald Duck' title='Donald Duck' />
```

CSS

```
img {  
  -webkit-filter: blur(1px);  
  filter: blur(1px);  
}
```

Result



Makes you wanna rub your glasses.

Read Filter Property online: <https://riptutorial.com/css/topic/1567/filter-property>

Chapter 24: Flexible Box Layout (Flexbox)

Introduction

The Flexible Box module, or just 'flexbox' for short, is a box model designed for user interfaces, and it allows users to align and distribute space among items in a container such that elements behave predictably when the page layout must accommodate different, unknown screen sizes. A flex container expands items to fill available space and shrinks them to prevent overflow.

Syntax

- `display: flex;`
- `flex-direction: row | row-reverse | column | column-reverse;`
- `flex-wrap: nowrap | wrap | wrap-reverse;`
- `flex-flow: <'flex-direction'> || <'flex-wrap'>`
- `justify-content: flex-start | flex-end | center | space-between | space-around;`
- `align-items: flex-start | flex-end | center | baseline | stretch;`
- `align-content: flex-start | flex-end | center | space-between | space-around | stretch;`
- `order: <integer>;`
- `flex-grow: <number>; /* default 0 */`
- `flex-shrink: <number>; /* default 1 */`
- `flex-basis: <length> | auto; /* default auto */`
- `flex: none | [ <'flex-grow'> <'flex-shrink'>? || <'flex-basis'> ]`
- `align-self: auto | flex-start | flex-end | center | baseline | stretch;`

Remarks

Vendor Prefixes

- `display: -webkit-box; /* Chrome <20 */`
- `display: -webkit-flex; /* Chrome 20+ */`
- `display: -moz-box; /* Firefox */`
- `display: -ms-flexbox; /* IE */`
- `display: flex; /* Modern browsers */`

Resources

- [A Complete Guide to Flexbox](#)
- [Solved by Flexbox](#)
- [What the Flexbox?!](#)
- [Flexbox in 5 minutes](#)
- [Flexbugs](#)

Examples

Sticky Variable-Height Footer

This code creates a sticky footer. When the content doesn't reach the end of the viewport, the footer sticks to the bottom of the viewport. When the content extends past the bottom of the viewport, the footer is also pushed out of the viewport. [View Result](#)

HTML:

```
<div class="header">
  <h2>Header</h2>
</div>

<div class="content">
  <h1>Content</h1>
  <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer nec odio. Praesent libero. Sed cursus ante dapibus diam. Sed nisi. Nulla quis sem at nibh elementum imperdiet. Duis sagittis ipsum. Praesent mauris. Fusce nec tellus sed augue semper porta. Mauris massa. Vestibulum lacinia arcu eget nulla. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Curabitur sodales ligula in libero. </p>
</div>

<div class="footer">
  <h4>Footer</h4>
</div>
```

CSS:

```
html, body {
  height: 100%;
}

body {
  display: flex;
  flex-direction: column;
}

.content {
  /* Include `0 auto` for best browser compatibility. */
  flex: 1 0 auto;
}

.header, .footer {
  background-color: grey;
  color: white;
  flex: none;
}
```

Holy Grail Layout using Flexbox

[Holy Grail layout](#) is a layout with a fixed height header and footer, and a center with 3 columns. The 3 columns include a fixed width sidenav, a fluid center, and a column for other content like ads (the fluid center appears first in the markup). CSS Flexbox can be used to achieve this with a

very simple markup:

HTML Markup:

```
<div class="container">
  <header class="header">Header</header>
  <div class="content-body">
    <main class="content">Content</main>
    <nav class="sidenav">Nav</nav>
    <aside class="ads">Ads</aside>
  </div>
  <footer class="footer">Footer</footer>
</div>
```

CSS:

```
body {
  margin: 0;
  padding: 0;
}

.container {
  display: flex;
  flex-direction: column;
  height: 100vh;
}

.header {
  flex: 0 0 50px;
}

.content-body {
  flex: 1 1 auto;

  display: flex;
  flex-direction: row;
}

.content-body .content {
  flex: 1 1 auto;
  overflow: auto;
}

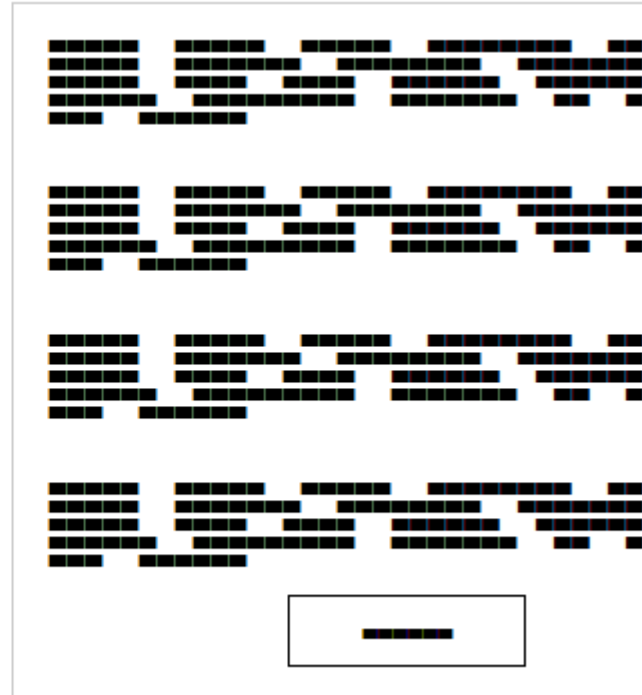
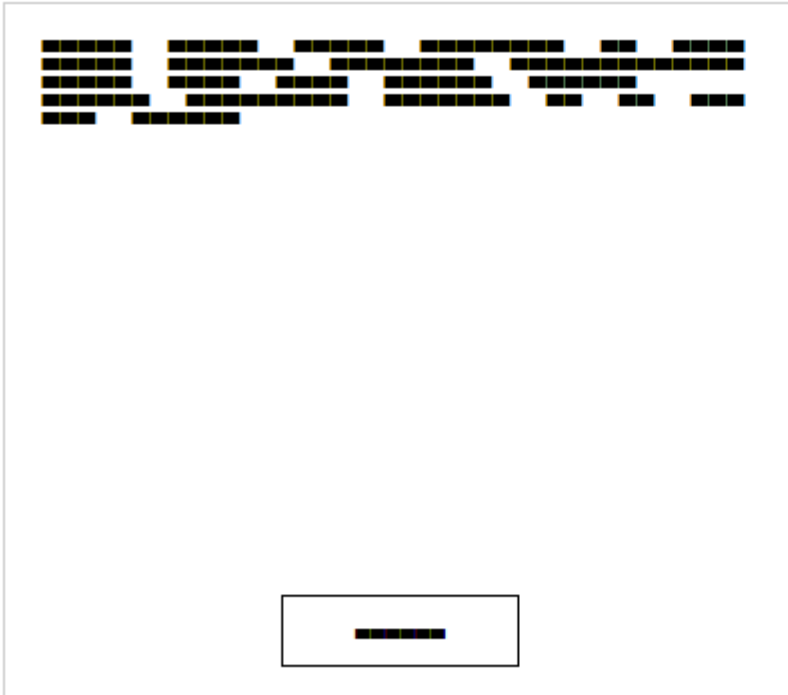
.content-body .sidenav {
  order: -1;
  flex: 0 0 100px;
  overflow: auto;
}

.content-body .ads {
  flex: 0 0 100px;
  overflow: auto;
}

.footer {
  flex: 0 0 50px;
}
```

Perfectly aligned buttons inside cards with flexbox

It's a regular pattern in design these days to vertically align **call to actions** inside its containing cards like this:



This can be achieved using a special trick with `flexbox`

HTML

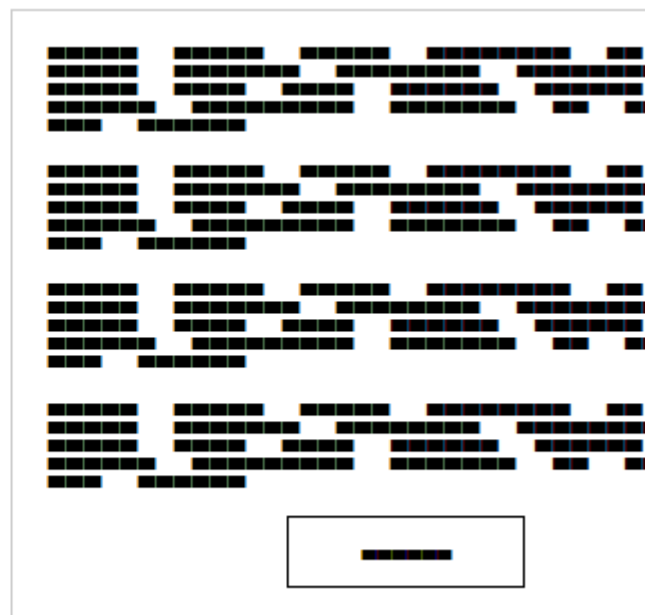
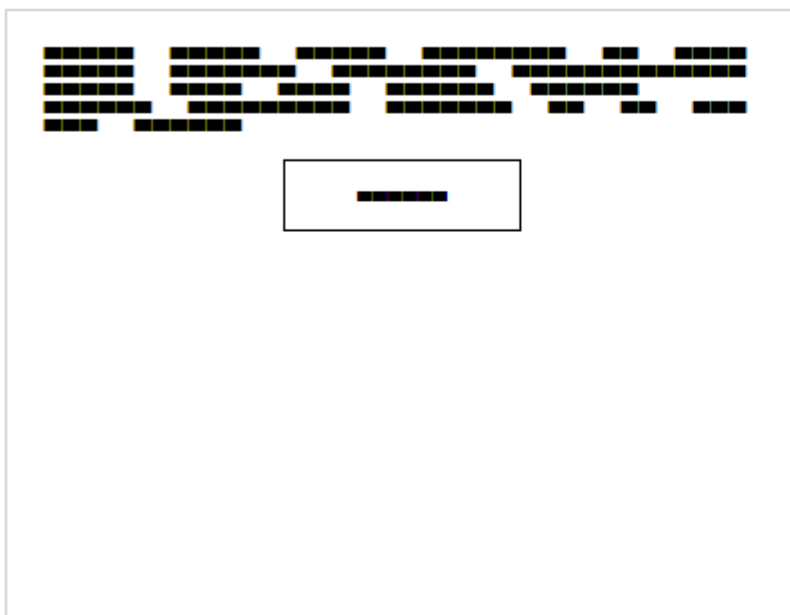
```
<div class="cards">
  <div class="card">
    <p>Lorem ipsum Magna proident ex anim dolor ullamco pariatur reprehenderit culpa esse enim
mollit labore dolore voluptate ullamco et ut sed qui minim.</p>
    <p><button>Action</button></p>
  </div>
  <div class="card">
    <p>Lorem ipsum Magna proident ex anim dolor ullamco pariatur reprehenderit culpa esse enim
mollit labore dolore voluptate ullamco et ut sed qui minim.</p>
    <p>Lorem ipsum Magna proident ex anim dolor ullamco pariatur reprehenderit culpa esse enim
mollit labore dolore voluptate ullamco et ut sed qui minim.</p>
    <p>Lorem ipsum Magna proident ex anim dolor ullamco pariatur reprehenderit culpa esse enim
mollit labore dolore voluptate ullamco et ut sed qui minim.</p>
    <p>Lorem ipsum Magna proident ex anim dolor ullamco pariatur reprehenderit culpa esse enim
mollit labore dolore voluptate ullamco et ut sed qui minim.</p>
    <p><button>Action</button></p>
  </div>
</div>
```

First of all, we use CSS to apply `display: flex;` to the container. This will create 2 columns equal in height with the content flowing naturally inside it

CSS

```
.cards {
  display: flex;
}
.card {
  border: 1px solid #ccc;
  margin: 10px 10px;
  padding: 0 20px;
}
button {
  height: 40px;
  background: #fff;
  padding: 0 40px;
  border: 1px solid #000;
}
p:last-child {
  text-align: center;
}
```

The layout will change and become like this:



In order to move the buttons to the bottom of the block, we need to apply `display: flex;` to the card itself with the direction set to `column`. After that, we should select the last element inside the card and set the `margin-top` to `auto`. This will push the last paragraph to the bottom of the card and achieve the required result.

Final CSS:

```
.cards {
  display: flex;
}
.card {
  border: 1px solid #ccc;
  margin: 10px 10px;
  padding: 0 20px;
}
```



```
display: flex;
flex-direction: column;
}
button {
  height: 40px;
  background: #fff;
  padding: 0 40px;
  border: 1px solid #000;
}
p:last-child {
  text-align: center;
  margin-top: auto;
}
```

Dynamic Vertical and Horizontal Centering (align-items, justify-content)

Simple Example (centering a single element)

HTML

```
<div class="aligner">
  <div class="aligner-item">...</div>
</div>
```

CSS

```
.aligner {
  display: flex;
  align-items: center;
  justify-content: center;
}

.aligner-item {
  max-width: 50%; /*for demo. Use actual width instead.*/
}
```

Here is a [demo](#).

Reasoning

Property	Value	Description
align-items	center	This centers the elements along the axis other than the one specified by <code>flex-direction</code> , i.e., vertical centering for a horizontal flexbox and horizontal centering for a vertical flexbox.
justify-content	center	This centers the elements along the axis specified by <code>flex-direction</code> .

Property	Value	Description
		I.e., for a horizontal (<code>flex-direction: row</code>) flexbox, this centers horizontally, and for a vertical flexbox (<code>flex-direction: column</code>) flexbox, this centers vertically)

Individual Property Examples

All of the below styles are applied onto this simple layout:

```
<div id="container">
  <div></div>
  <div></div>
  <div></div>
</div>
```

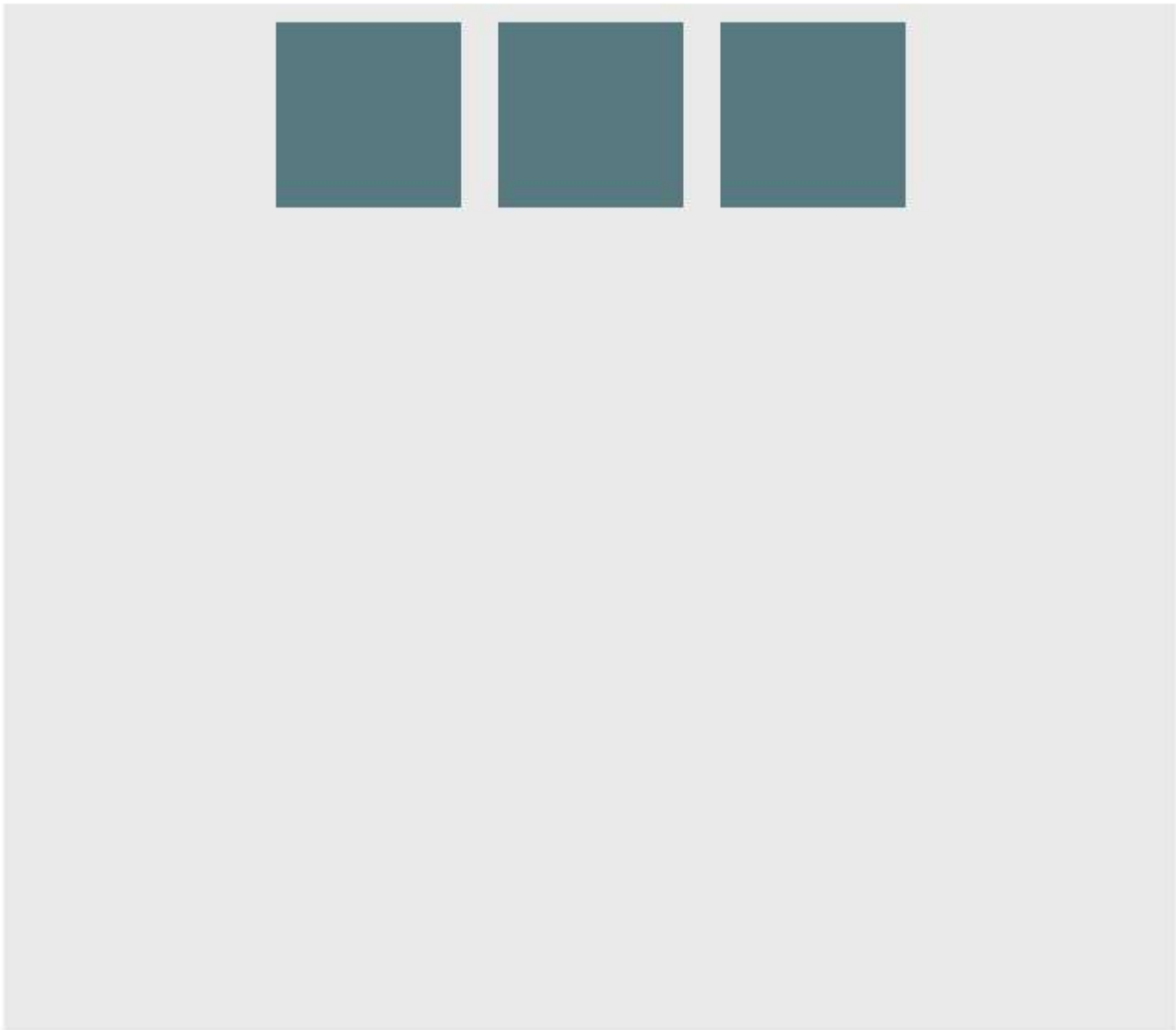
where `#container` is the `flex-box`.

Example: `justify-content: center` on a horizontal flexbox

CSS:

```
div#container {
  display: flex;
  flex-direction: row;
  justify-content: center;
}
```

Outcome:



Here is a [demo](#).

Example: `justify-content: center` on a vertical flexbox

CSS:

```
div#container {  
  display: flex;  
  flex-direction: column;  
  justify-content: center;  
}
```

Outcome:



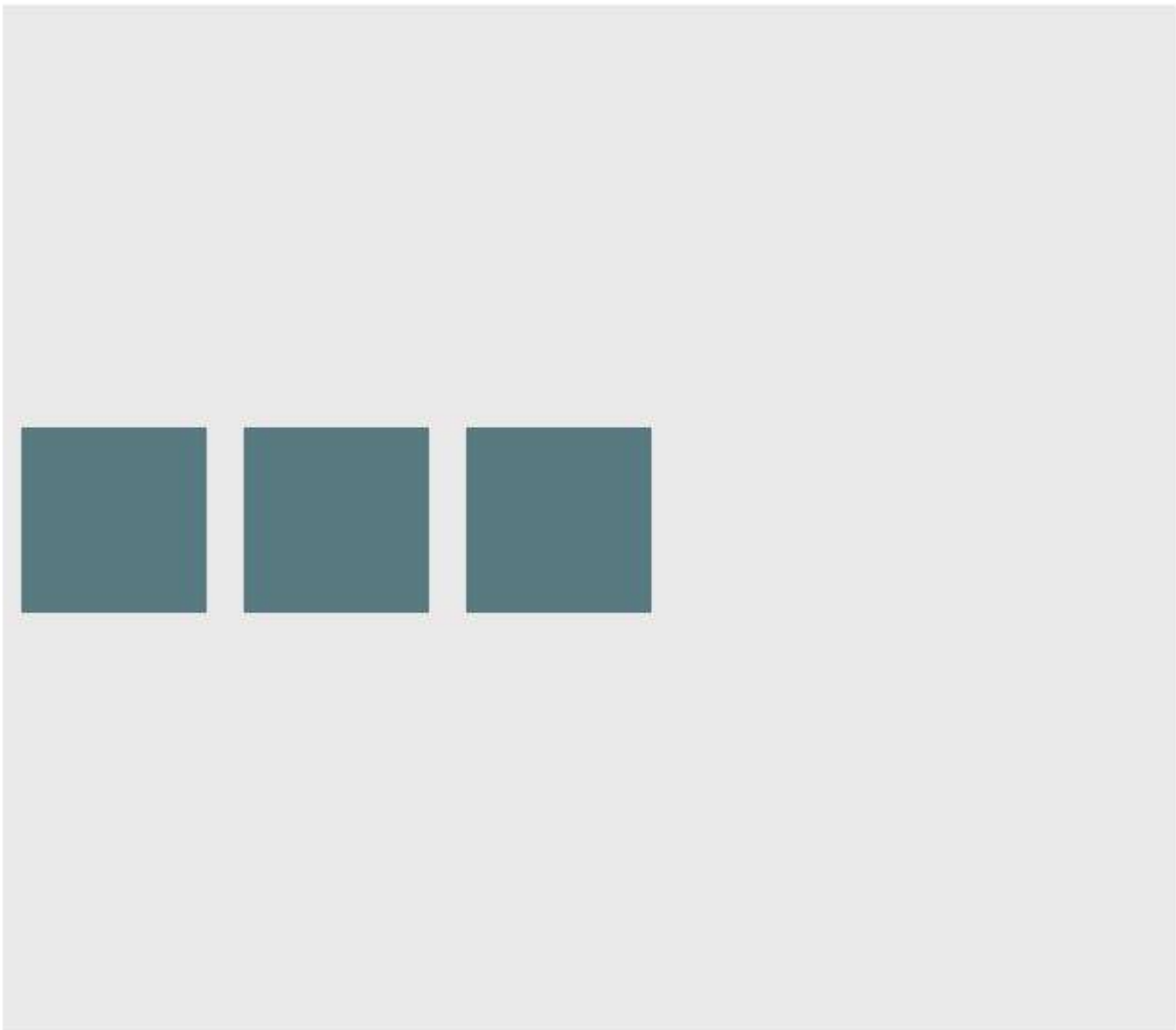
Here is a [demo](#).

Example: `align-content: center` on a horizontal flexbox

CSS:

```
div#container {  
  display: flex;  
  flex-direction: row;  
  align-items: center;  
}
```

Outcome:



Here is a [demo](#).

Example: `align-content: center` on a vertical flexbox

CSS:

```
div#container {  
  display: flex;  
  flex-direction: column;  
  align-items: center;  
}
```

Outcome:

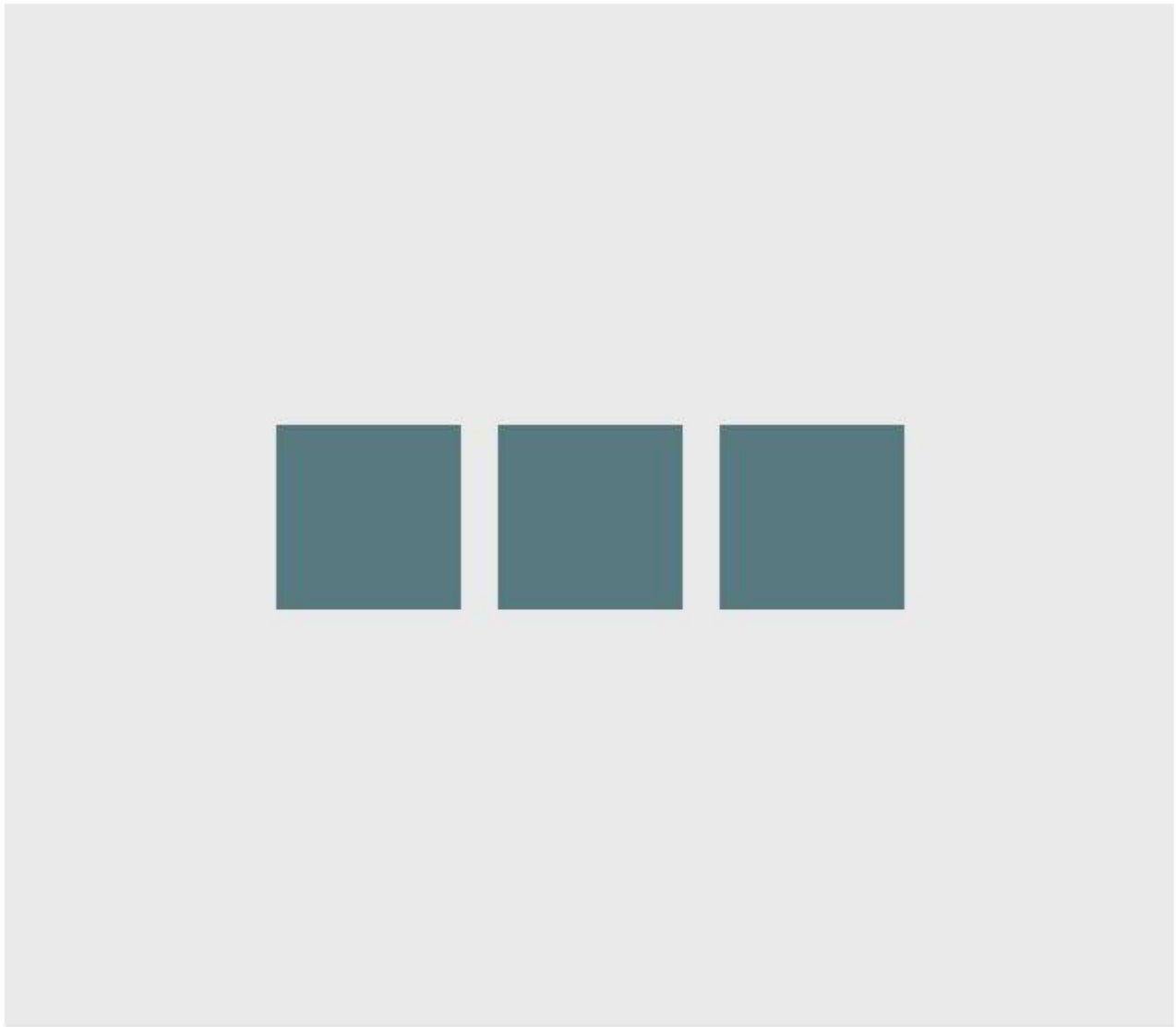


Here is a [demo](#).

Example: Combination for centering both on horizontal flexbox

```
div#container {  
  display: flex;  
  flex-direction: row;  
  justify-content: center;  
  align-items: center;  
}
```

Outcome:



Here is a [demo](#).

Example: Combination for centering both on vertical flexbox

```
div#container {  
  display: flex;  
  flex-direction: column;  
  justify-content: center;  
  align-items: center;  
}
```

Outcome:



Here is a [demo](#).

Same height on nested containers

This code makes sure that all nested containers are always the same height. This is done by assuring that all nested elements are the same height as the containing parent div. [See working example](#): <https://jsfiddle.net/3wwh7ewp/>

This effect is achieved due to the property `align-items` being set to `stretch` by default.

HTML

```
<div class="container">
  <div style="background-color: red">
    Some <br />
    data <br />
    to make<br />
    a height <br />
  </div>
  <div style="background-color: blue">
```



```
Fewer <br />
lines <br />
</div>
</div>
```

CSS

```
.container {
  display: flex;
  align-items: stretch; // Default value
}
```

Note: [Does not work on IE versions under 10](#)

Optimally fit elements to their container

One of the nicest features of flexbox is to allow optimally fitting containers to their parent element.

[Live demo.](#)

HTML:

```
<div class="flex-container">
  <div class="flex-item">1</div>
  <div class="flex-item">2</div>
  <div class="flex-item">3</div>
  <div class="flex-item">4</div>
  <div class="flex-item">5</div>
</div>
```

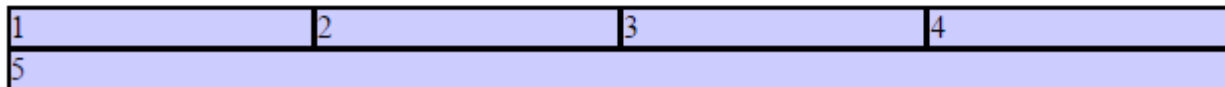
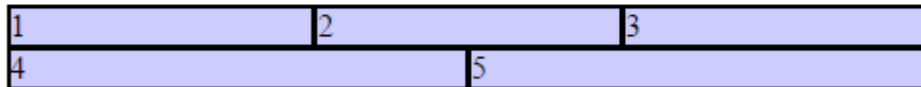
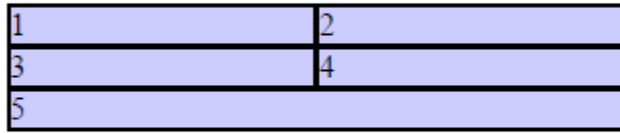
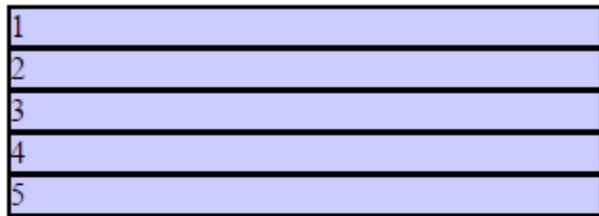
CSS:

```
.flex-container {
  background-color: #000;
  height: 100%;
  display: flex;
  flex-direction: row;
  flex-wrap: wrap;
  justify-content: flex-start;
  align-content: stretch;
  align-items: stretch;
}

.flex-item {
  background-color: #ccf;
  margin: 0.1em;
  flex-grow: 1;
  flex-shrink: 0;
  flex-basis: 200px; /* or % could be used to ensure a specific layout */
}
```

Outcome:

Columns adapt as screen is resized.



Read Flexible Box Layout (Flexbox) online: <https://riptutorial.com/css/topic/445/flexible-box-layout-flexbox->

Chapter 25: Floats

Syntax

- clear: none | left | right | both | inline-start | inline-end;
- float: left | right | none | inline-start | inline-end;

Remarks

As float implies the use of the block layout, it modifies the computed value of the display values in some cases [1]

[1]: <https://developer.mozilla.org/en-US/docs/Web/CSS/float> MDN

Examples

Float an Image Within Text

The most basic use of a float is having text wrap around an image. The below code will produce two paragraphs and an image, with the second paragraph flowing around the image. Notice that it is always content *after* the floated element that flows around the floated element.

HTML:

```
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer nec odio. Praesent libero. Sed cursus ante dapibus diam. Sed nisi. Nulla quis sem at nibh elementum imperdiet. Duis sagittis ipsum. Praesent mauris. Fusce nec tellus sed augue semper porta. Mauris massa. Vestibulum lacinia arcu eget nulla. </p>



<p>Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Curabitur sodales ligula in libero. Sed dignissim lacinia nunc. Curabitur tortor. Pellentesque nibh. Aenean quam. In scelerisque sem at dolor. Maecenas mattis. Sed convallis tristique sem. Proin ut ligula vel nunc egestas porttitor. Morbi lectus risus, iaculis vel, suscipit quis, luctus non, massa. Fusce ac turpis quis ligula lacinia aliquet. </p>
```

CSS:

```
img {
  float:left;
  margin-right:1rem;
}
```

This will be the output

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer nec odio. Praesent libero. Sed cursus ante dapibus diam. Sed nisi. Nulla quis sem at nibh elementum imperdiet. Duis sagittis ipsum. Praesent mauris. Fusce nec tellus sed augue semper porta. Mauris massa. Vestibulum lacinia arcu eget nulla.



Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Curabitur sodales ligula in libero.

Sed dignissim lacinia nunc. Curabitur tortor. Pellentesque nibh. Aenean quam. In scelerisque sem at dolor. Maecenas mattis. Sed convallis tristique sem. Proin ut ligula vel nunc egestas porttitor. Morbi lectus risus, iaculis vel, suscipit quis, luctus non, massa. Fusce ac turpis quis ligula lacinia aliquet.

[Codepen Link](#)

Simple Two Fixed-Width Column Layout

A simple two-column layout consists of two fixed-width, floated elements. Note that the sidebar and content area are not the same height in this example. This is one of the tricky parts with multi-column layouts using floats, and requires workarounds to make multiple columns appear to be the same height.

HTML:

```

<div class="wrapper">

<div class="sidebar">
  <h2>Sidebar</h2>

  <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer nec odio.</p>
</div>

<div class="content">
  <h1>Content</h1>

  <p>Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos
himenaeos. Curabitur sodales ligula in libero. Sed dignissim lacinia nunc. Curabitur tortor.
Pellentesque nibh. Aenean quam. In scelerisque sem at dolor. Maecenas mattis. Sed convallis
tristique sem. Proin ut ligula vel nunc egestas porttitor. Morbi lectus risus, iaculis vel,
suscipit quis, luctus non, massa. Fusce ac turpis quis ligula lacinia aliquet. </p>
</div>

</div>

```

CSS:

```

.wrapper {
  width:600px;
  padding:20px;
  background-color:pink;

  /* Floated elements don't use any height. Adding "overflow:hidden;" forces the
  parent element to expand to contain its floated children. */
  overflow:hidden;
}

.sidebar {
  width:150px;
  float:left;
  background-color:blue;
}

.content {
  width:450px;
  float:right;
  background-color:yellow;
}

```

Simple Three Fixed-Width Column Layout

HTML:

```

<div class="wrapper">
  <div class="left-sidebar">
    <h1>Left Sidebar</h1>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. </p>
  </div>
  <div class="content">
    <h1>Content</h1>
    <p>Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos
himenaeos. Curabitur sodales ligula in libero. Sed dignissim lacinia nunc. Curabitur tortor.
Pellentesque nibh. Aenean quam. In scelerisque sem at dolor. Maecenas mattis. Sed convallis

```

```

tristique sem. Proin ut ligula vel nunc egestas porttitor. Morbi lectus risus, iaculis vel,
suscipit quis, luctus non, massa. </p>
</div>
<div class="right-sidebar">
  <h1>Right Sidebar</h1>
  <p>Fusce ac turpis quis ligula lacinia aliquet.</p>
</div>
</div>

```

CSS:

```

.wrapper {
  width:600px;
  background-color:pink;
  padding:20px;

  /* Floated elements don't use any height. Adding "overflow:hidden;" forces the
  parent element to expand to contain its floated children. */
  overflow:hidden;
}

.left-sidebar {
  width:150px;
  background-color:blue;
  float:left;
}

.content {
  width:300px;
  background-color:yellow;
  float:left;
}

.right-sidebar {
  width:150px;
  background-color:green;
  float:right;
}

```

Two-Column Lazy/Greedy Layout

This layout uses one floated column to create a two-column layout with no defined widths. In this example the left sidebar is "lazy," in that it only takes up as much space as it needs. Another way to say this is that the left sidebar is "shrink-wrapped." The right content column is "greedy," in that it takes up all the remaining space.

HTML:

```

<div class="sidebar">
  <h1>Sidebar</h1>
  
</div>

<div class="content">
  <h1>Content</h1>
  <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer nec odio. Praesent libero.

```

```
Sed cursus ante dapibus diam. Sed nisi. Nulla quis sem at nibh elementum imperdiet. Duis sagittis ipsum. Praesent mauris. Fusce nec tellus sed augue semper porta. Mauris massa. Vestibulum lacinia arcu eget nulla. </p>
<p>Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Curabitur sodales ligula in libero. Sed dignissim lacinia nunc. Curabitur tortor. Pellentesque nibh. Aenean quam. In scelerisque sem at dolor. Maecenas mattis. Sed convallis tristique sem. Proin ut ligula vel nunc egestas porttitor. Morbi lectus risus, iaculis vel, suscipit quis, luctus non, massa. Fusce ac turpis quis ligula lacinia aliquet. Mauris ipsum. Nulla metus metus, ullamcorper vel, tincidunt sed, euismod in, nibh. </p>
</div>
```

CSS:

```
.sidebar {
  /* `display:table;` shrink-wraps the column */
  display:table;
  float:left;
  background-color:blue;
}

.content {
  /* `overflow:hidden;` prevents `.content` from flowing under `.sidebar` */
  overflow:hidden;
  background-color:yellow;
}
```

Fiddle

clear property

The clear property is directly related to floats. Property Values:

- none - Default. Allows floating elements on both sides
- left - No floating elements allowed on the left side
- right - No floating elements allowed on the right side
- both - No floating elements allowed on either the left or the right side
- initial - Sets this property to its default value. Read about initial
- inherit - Inherits this property from its parent element. Read about inherit

```
<html>
<head>
<style>
img {
  float: left;
}

p.clear {
  clear: both;
}
</style>
</head>
<body>


<p>Lorem ipsum Lorem ipsum Lorem ipsum Lorem ipsum Lorem ipsum Lorem ipsum Lorem ipsum
```

```

Lorem ipsum Lorem ipsum Lorem ipsum Lorem ipsum Lorem ipsum </p>
<p class="clear">Lorem ipsum Lorem ipsum Lorem ipsum Lorem ipsum Lorem ipsum Lorem ipsum Lorem ipsum
Lorem ipsum Lorem ipsum Lorem ipsum Lorem ipsum Lorem ipsum Lorem ipsum Lorem ipsum </p>

</body>
</html>

```

Clearfix

The clearfix hack is a popular way to contain floats (N. Gallagher aka @necolas)

Not to be confused with the `clear` property, clearfix is a *concept* (that is also related to floats, thus the possible confusion). To *contain floats*, you've to add `.cf` or `.clearfix` class on the container (**the parent**) and style this class with a few rules described below.

3 versions with slightly different effects (sources :[A new micro clearfix hack](#) by N. Gallagher and [clearfix reloaded](#) by T. J. Koblenz):

Clearfix (with top margin collapsing of contained floats still occurring)

```

.cf:after {
    content: "";
    display: table;
}

.cf:after {
    clear: both;
}

```

Clearfix also preventing top margin collapsing of contained floats

```

/**
 * For modern browsers
 * 1. The space content is one way to avoid an Opera bug when the
 *    contenteditable attribute is included anywhere else in the document.
 *    Otherwise it causes space to appear at the top and bottom of elements
 *    that are clearfixed.
 * 2. The use of `table` rather than `block` is only necessary if using
 *    `:before` to contain the top-margins of child elements.
 */
.cf:before,
.cf:after {
    content: " "; /* 1 */
    display: table; /* 2 */
}

```



```
.cf:after {  
    clear: both;  
}
```

Clearfix with support of outdated browsers IE6 and IE7

```
.cf:before,  
.cf:after {  
    content: " ";  
    display: table;  
}  
  
.cf:after {  
    clear: both;  
}  
  
/**  
 * For IE 6/7 only  
 * Include this rule to trigger hasLayout and contain floats.  
 */  
.cf {  
    *zoom: 1;  
}
```

[Codepen showing clearfix effect](#)

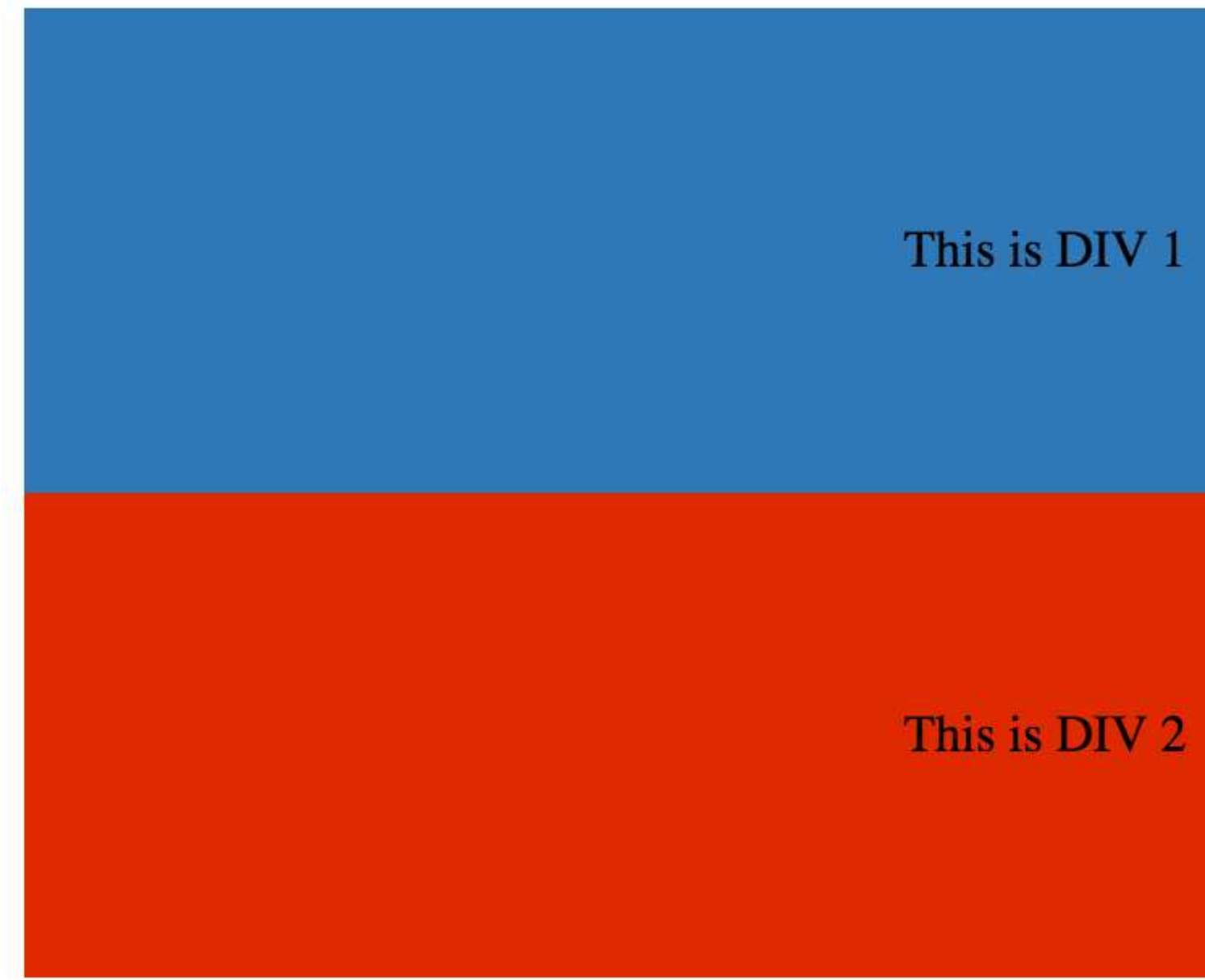
Other resource: [Everything you know about clearfix is wrong](#) (clearfix and BFC - Block Formatting Context while hasLayout relates to outdated browsers IE6 maybe 7)

In-line DIV using float

The `div` is a block-level element, i.e it occupies the whole of the page width and the siblings are place one below the other irrespective of their width.

```
<div>  
    <p>This is DIV 1</p>  
</div>  
<div>  
    <p>This is DIV 2</p>  
</div>
```

The output of the following code will be



This is DIV 1

This is DIV 2

We can make them in-line by adding a `float` css property to the `div`.

HTML:

```
<div class="outer-div">
  <div class="inner-div1">
    <p>This is DIV 1</p>
  </div>
  <div class="inner-div2">
    <p>This is DIV 2</p>
  </div>
</div>
```

CSS

```
.inner-div1 {
  width: 50%;
  margin-right: 0px;
  float: left;
```

```
background : #337ab7;
padding:50px 0px;
}

.inner-div2 {
width: 50%;
margin-right:0px;
float:left;
background : #dd2c00;
padding:50px 0px;
}

p {
text-align:center;
}
```



This is DIV 1

[Codepen Link](#)

Use of overflow property to clear floats

Setting `overflow` value to `hidden`, `auto` or `scroll` to an element, will clear all the floats within that element.

Note: using `overflow:scroll` will always show the scrollbar

Read Floats online: <https://riptutorial.com/css/topic/405/floats>

Chapter 26: Fragmentation

Syntax

- `page-break-after`: `auto` | `always` | `avoid` | `left` | `right` | `initial` | `inherit`;
- `page-break-before`: `auto` | `always` | `avoid` | `left` | `right` | `initial` | `inherit`;
- `page-break-inside`: `auto` | `avoid` | `initial` | `inherit`;

Parameters

Value	Description
auto	Default. Automatic page breaks
always	Always insert a page break
avoid	Avoid page break (if possible)
left	Insert page breaks so that the next page is formatted as a left page
right	Insert page breaks so that the next page is formatted as a right page
initial	Sets this property to its default value.
inherit	Inherits this property from its parent element.

Remarks

There is no `page-break` property in CSS. Only the 3 properties (**`page-break-before`**, **`page-break-after`**, **`page-break-inside`**).

Related: `orphans`, `widows`.

Examples

Media print page-break

```
@media print {
  p {
    page-break-inside: avoid;
  }
  h1 {
    page-break-before: always;
  }
  h2 {
    page-break-after: avoid;
  }
}
```

```
}  
}
```

This code does 3 things:

- it prevents a page break inside any p tags, meaning a paragraph will never be broken in two pages, if possible.
- it forces a page-break-before in all h1 headings, meaning that before every h1 occurrence, there will be a page break.
- it prevents page-breaks right after any h2

Read Fragmentation online: <https://riptutorial.com/css/topic/4316/fragmentation>

Chapter 27: Functions

Syntax

- `<calc()> = calc( <calc-sum> )`
- `<calc-sum> = <calc-product> [ [ '+' | '-' ] <calc-product> ]*`
- `<calc-product> = <calc-value> [ '*' <calc-value> | '/' <number> ]*`
- `<calc-value> = <number> | <dimension> | <percentage> | ( <calc-sum> )`

Remarks

For `calc()`, white space is required around the "-" and "+" operators, but not the "*" or "/" operators.

All units must be of the same type; trying to multiply a height by a time duration, for example, is invalid.

Examples

`calc()` function

Accepts a mathematical expression and returns a numerical value.

It is especially useful when working with different types of units (e.g. subtracting a px value from a percentage) to calculate the value of an attribute.

+, -, /, and * operators can all be used, and parentheses can be added to specify the order of operations if necessary.

Use `calc()` to calculate the width of a div element:

```
#div1 {
  position: absolute;
  left: 50px;
  width: calc(100% - 100px);
  border: 1px solid black;
  background-color: yellow;
  padding: 5px;
  text-align: center;
}
```

Use `calc()` to determine the position of a background-image:

```
background-position: calc(50% + 17px) calc(50% + 10px), 50% 50%;
```

Use `calc()` to determine the height of an element:

```
height: calc(100% - 20px);
```

attr() function

Returns the value of an attribute of the selected element.

Below is a blockquote element which contains a character inside a `data-* attribute` which CSS can use (e.g. inside the `::before` and `::after` [pseudo-element](#)) using this function.

```
<blockquote data-mark='"'></blockquote>
```

In the following CSS block, the character is appended before and after the text inside the element:

```
blockquote[data-mark]::before,  
blockquote[data-mark]::after {  
    content: attr(data-mark);  
}
```

linear-gradient() function

Creates a image representing a linear gradient of colors.

```
linear-gradient( 0deg, red, yellow 50%, blue);
```

This creates a gradient going from bottom to top, with colors starting at red, then yellow at 50%, and finishing in blue.

radial-gradient() function

Creates an image representing a gradient of colors radiating from the center of the gradient

```
radial-gradient(red, orange, yellow) /*A gradient coming out from the middle of the  
gradient, red at the center, then orange, until it is finally yellow at the edges*/
```

var() function

The `var()` function allows CSS variables to be accessed.

```
/* set a variable */  
:root {  
    --primary-color: blue;  
}  
  
/* access variable */  
selector {  
    color: var(--primary-color);  
}
```

This feature is currently under development. Check caniuse.com for the latest browser support.

Read Functions online: <https://riptutorial.com/css/topic/2214/functions>

Chapter 28: Grid

Introduction

Grid layout is a new and powerful CSS layout system that allows to divide a web page content into rows and columns in an easy way.

Remarks

[CSS Grid Layout Module Level 1](#) is, as of 9 September 2016, a W3C Candidate Recommendation. It is considered to be in the Testing stage (<https://www.w3.org/Style/CSS/current-work>).

As of 3 July 2017, Microsoft's Internet Explorer 10 and 11 and Edge browsers only support an older version of the specification using a vendor prefix.

Examples

Basic Example

Property	Possible Values
display	grid / inline-grid

The CSS Grid is defined as a display property. It applies to a parent element and its immediate children only.

Consider the following markup:

```
<section class="container">
  <div class="item1">item1</div>
  <div class="item2">item2</div>
  <div class="item3">item3</div>
  <div class="item4">item4</div>
</section>
```

The easiest way to define the markup structure above as a grid is to simply set its `display` property to `grid`:

```
.container {
  display: grid;
}
```

However, doing this will invariably cause all the child elements to collapse on top of one another. This is because the children do not currently know how to position themselves within the grid. But we can explicitly tell them.

First we need to tell the grid element `.container` how many rows and columns will make up its structure and we can do this using the `grid-columns` and `grid-rows` properties (note the pluralisation):

```
.container {  
  display: grid;  
  grid-columns: 50px 50px 50px;  
  grid-rows: 50px 50px;  
}
```

However, that still doesn't help us much because we need to give an order to each child element. We can do this by specifying the `grid-row` and `grid-column` values which will tell it where it sits in the grid:

```
.container .item1 {  
  grid-column: 1;  
  grid-row: 1;  
}  
.container .item2 {  
  grid-column: 2;  
  grid-row: 1;  
}  
.container .item3 {  
  grid-column: 1;  
  grid-row: 2;  
}  
.container .item4 {  
  grid-column: 2;  
  grid-row: 2;  
}
```

By giving each item a column and row value it identifies the items order within the container.

View a working example on [JSFiddle](#). You'll need to view this in IE10, IE11 or Edge for it to work as these are currently the only browsers supporting Grid Layout (with vendor prefix `-ms-`) or enable a flag in Chrome, Opera and Firefox according to [caniuse](#) in order to test with them.

Read Grid online: <https://riptutorial.com/css/topic/2152/grid>

Chapter 29: Inheritance

Syntax

- *property*: inherit;

Examples

Automatic inheritance

Inheritance is a fundamental mechanism of CSS by which the computed values of some properties of an element are applied to its' children. This is particularly useful when you want to set a global style to your elements rather than having to set said properties to each and every element in your markup.

Common properties that are automatically inherited are: `font`, `color`, `text-align`, `line-height`.

Assume the following stylesheet:

```
#myContainer {
  color: red;
  padding: 5px;
}
```

This will apply `color: red` not only to the `<div>` element but also to the `<h3>` and `<p>` elements. However, due to the nature of `padding` its value will **not** be inherited to those elements.

```
<div id="myContainer">
  <h3>Some header</h3>
  <p>Some paragraph</p>
</div>
```

Enforced inheritance

Some properties are not automatically inherited from an element down to its' children. This is because those properties are typically desired to be unique to the element (or selection of elements) to which the property is applied to. Common such properties are `margin`, `padding`, `background`, `display`, etc.

However, sometimes inheritance is desired anyway. To achieve this, we can apply the `inherit` value to the property that should be inherited. The `inherit` value can be applied to *any* CSS property and *any* HTML element.

Assume the following stylesheet:

```
#myContainer {
  color: red;
```

```
padding: 5px;
}
#myContainer p {
padding: inherit;
}
```

This will apply `color: red` to both the `<h3>` and `<p>` elements due to the inheritance nature of the `color` property. However, the `<p>` element will also inherit the `padding` value from its' parent because this was specified.

```
<div id="myContainer">
  <h3>Some header</h3>
  <p>Some paragraph</p>
</div>
```

Read Inheritance online: <https://riptutorial.com/css/topic/3586/inheritance>

Chapter 30: Inline-Block Layout

Examples

Justified navigation bar

The horizontally justified navigation (menu) bar has some number of items that should be justified. The first (left) item has no left margin within the container, the last (right) item has no right margin within the container. The distance between items is equal, independent on the individual item width.

HTML

```
<nav>
  <ul>
    <li>abc</li>
    <li>abcdefghijkl</li>
    <li>abcdef</li>
  </ul>
</nav>
```

CSS

```
nav {
  width: 100%;
  line-height: 1.4em;
}
ul {
  list-style: none;
  display: block;
  width: 100%;
  margin: 0;
  padding: 0;
  text-align: justify;
  margin-bottom: -1.4em;
}
ul:after {
  content: "";
  display: inline-block;
  width: 100%;
}
li {
  display: inline-block;
}
```

Notes

- The `nav`, `ul` and `li` tags were chosen for their semantic meaning of 'a list of navigation (menu) items'. Other tags may also be used of course.
- The `:after` pseudo-element causes an extra 'line' in the `ul` and thus an extra, empty height of this block, pushing other content down. This is solved by the negative `margin-bottom`, which has to have the same magnitude as the `line-height` (but negative).
- If the page becomes too narrow for all the items to fit, the items will break to a new line (starting from the right) and be justified on this line. The total height of the menu will grow as needed.

Read Inline-Block Layout online: <https://riptutorial.com/css/topic/3308/inline-block-layout>

Chapter 31: Internet Explorer Hacks

Remarks

These “hacks” may be used to target a specific browser/client. This may be used to work around browser rendering differences by applying styles in one of those wrappers listed above.

Examples

High Contrast Mode in Internet Explorer 10 and greater

In Internet Explorer 10+ and Edge, Microsoft provides the `-ms-high-contrast` media selector to expose the "High Contrast" setting from the browser, which allows the programmer to adjust their site's styles accordingly.

The `-ms-high-contrast` selector has 3 states: `active`, `black-on-white`, and `white-on-black`. In IE10+ it also had a `none` state, but that is no longer supported in Edge going forward.

Examples

```
@media screen and (-ms-high-contrast: active), (-ms-high-contrast: black-on-white) {
    .header{
        background: #fff;
        color: #000;
    }
}
```

This will change the header background to white and the text color to black when high contrast mode is active *and* it is in `black-on-white` mode.

```
@media screen and (-ms-high-contrast: white-on-black) {
    .header{
        background: #000;
        color: #fff;
    }
}
```

Similar to the first example, but this specifically selects the `white-on-black` state only, and inverts the header colors to a black background with white text.

More Information:

[Microsoft Documentation](#) on `-ms-high-contrast`

Internet Explorer 6 & Internet Explorer 7 only

To target Internet Explorer 6 and Internet Explorer 7, start your properties with `*`:

```
.hide-on-ie6-and-ie7 {  
    *display : none; // This line is processed only on IE6 and IE7  
}
```

Non-alphanumeric prefixes (other than hyphens and underscores) are ignored in IE6 and IE7, so this hack works for any unprefixed `property: value` pair.

Internet Explorer 8 only

To target Internet Explorer 8, wrap your selectors inside `@media \0 screen { }`:

```
@media \0 screen {  
    .hide-on-ie8 {  
        display : none;  
    }  
}
```

Everything between `@media \0 screen { }` is processed only by I

Adding Inline Block support to IE6 and IE7

```
display: inline-block;
```

The `display` property with the value of `inline-block` is not supported by Internet Explorer 6 and 7. A work-around for this is:

```
zoom: 1;  
*display: inline;
```

The `zoom` property triggers the `hasLayout` feature of elements, and it is available only in Internet Explorer. The `*display` makes sure that the invalid property executes only on the affected browsers. Other browsers will simply ignore the rule.

Read Internet Explorer Hacks online: <https://riptutorial.com/css/topic/5056/internet-explorer-hacks>

Chapter 32: Layout Control

Syntax

- display: none | inline | block | list-item | inline-list-item | inline-block | inline-table | table | table-cell | table-column | table-column-group | table-footer-group | table-header-group | table-row | table-row-group | flex | inline-flex | grid | inline-grid | run-in | ruby | ruby-base | ruby-text | ruby-base-container | ruby-text-container | contents;

Parameters

Value	Effect
none	Hide the element and prevent it from occupying space.
block	Block element, occupy 100% of the available width, break after element.
inline	Inline element, occupy no width, no break after element.
inline-block	Taking special properties from both inline and block elements, no break, but can have width.
inline-flex	Displays an element as an inline-level flex container.
inline-table	The element is displayed as an inline-level table.
grid	Behaves like a block element and lays out its content according to the grid model.
flex	Behaves like a block element and lays out its content according to the flexbox model.
inherit	Inherit the value from the parent element.
initial	Reset the value to the default value taken from behaviors described in the HTML specifications or from the browser/user default stylesheet.
table	Behaves like the HTML <code>table</code> element.
table-cell	Let the element behave like a <code><td></code> element
table-column	Let the element behave like a <code><col></code> element
table-row	Let the element behave like a <code><tr></code> element
list-item	Let the element behave like a <code></code> element.

Examples

The display property

The `display` CSS property is fundamental for controlling the layout and flow of an HTML document. Most elements have a default `display` value of either `block` or `inline` (though some elements have other default values).

Inline

An `inline` element occupies only as much width as necessary. It stacks horizontally with other elements of the same type and may not contain other non-inline elements.

```
<span>This is some <b>bolded</b> text!</span>
```

This is some **bolded** text!

As demonstrated above, two `inline` elements, `<span>` and `<b>`, are in-line (hence the name) and do not break the flow of the text.

Block

A `block` element occupies the maximum available width of its' parent element. It starts with a new line and, in contrast to `inline` elements, it does not restrict the type of elements it may contain.

```
<div>Hello world!</div><div>This is an example!</div>
```

Hello world!
This is an example!

The `div` element is block-level by default, and as shown above, the two `block` elements are vertically stacked and, unlike the `inline` elements, the flow of the text breaks.

Inline Block

The `inline-block` value gives us the best of both worlds: it blends the element in with the flow of the text while allowing us to use `padding`, `margin`, `height` and similar properties which have no visible effect on `inline` elements.

Elements with this display value act as if they were regular text and as a result are affected by rules controlling the flow of text such as `text-align`. By default they are also shrunk to the the smallest size possible to accommodate their content.

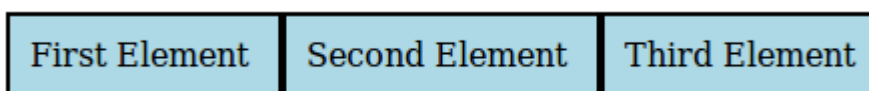
```

<!--Inline: unordered list-->
<style>
li {
    display : inline;
    background : lightblue;
    padding:10px;

    border-width:2px;
    border-color:black;
    border-style:solid;
}
</style>

<ul>
<li>First Element </li>
<li>Second Element </li>
<li>Third Element </li>
</ul>

```



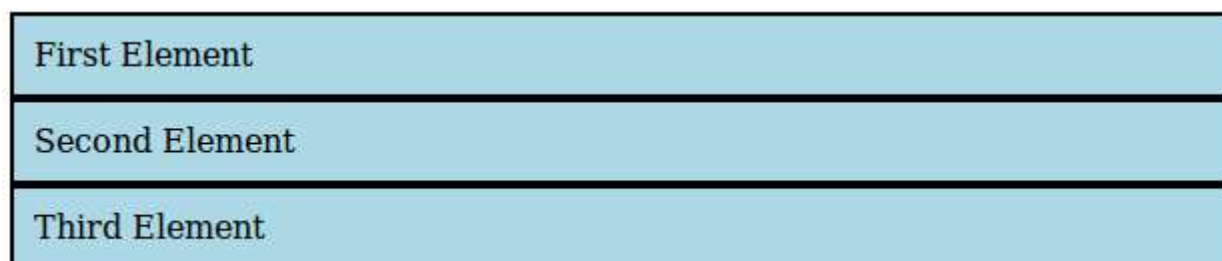
```

<!--block: unordered list-->
<style>
li {
    display : block;
    background : lightblue;
    padding:10px;

    border-width:2px;
    border-color:black;
    border-style:solid;
}
</style>

<ul>
<li>First Element </li>
<li>Second Element </li>
<li>Third Element </li>
</ul>

```



```

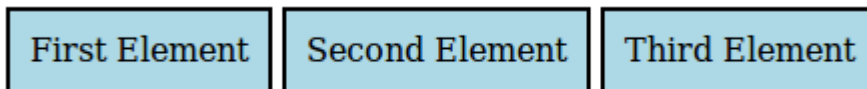
<!--Inline-block: unordered list-->
<style>
li {
    display : inline-block;
    background : lightblue;
    padding:10px;

```

```
border-width:2px;
border-color:black;
border-style:solid;
}

</style>

<ul>
<li>First Element </li>
<li>Second Element </li>
<li>Third Element </li>
</ul>
```



none

An element that is given the none value to its display property will not be displayed at all.

For example let's create a div-element that has an id of `myDiv`:

```
<div id="myDiv"></div>
```

This can now be marked as not being displayed by the following CSS rule:

```
#myDiv {
  display: none;
}
```

When an element has been set to be `display:none`; the browser ignores every other layout property for that specific element (both `position` and `float`). No box will be rendered for that element and its existence in html does not affect the position of following elements.

Note that this is different from setting the `visibility` property to `hidden`. Setting `visibility: hidden`; for an element would not display the element on the page but the element would still take up the space in the rendering process as if it would be visible. This will therefore affect how following elements are displayed on the page.

The `none` value for the display property is commonly used along with JavaScript to show or hide elements at will, eliminating the need to actually delete and re-create them.

To get old table structure using div

This is the normal HTML table structure

```
<style>
  table {
    width: 100%;
```

```
    }
</style>

<table>
  <tr>
    <td>
      I'm a table
    </td>
  </tr>
</table>
```

You can do same implementation like this

```
<style>
  .table-div {
    display: table;
  }
  .table-row-div {
    display: table-row;
  }
  .table-cell-div {
    display: table-cell;
  }
</style>

<div class="table-div">
  <div class="table-row-div">
    <div class="table-cell-div">
      I behave like a table now
    </div>
  </div>
</div>
```

Read Layout Control online: <https://riptutorial.com/css/topic/1473/layout-control>

Chapter 33: Length Units

Introduction

A CSS distance measurement is a number immediately followed by a length unit (px, em, pc, in, ...)

CSS supports a number of length measurements units. They are absolute or relative.

Syntax

- *value***unit**
- 1em

Parameters

Unit	Description
%	Define sizes in terms of parent objects or current object dependent on property
em	Relative to the font-size of the element (2em means 2 times the size of the current font)
rem	Relative to font-size of the root element
vw	Relative to 1% of the width of the viewport*
vh	Relative to 1% of the height of the viewport*
vmin	Relative to 1% of viewport's* smaller dimension
vmax	Relative to 1% of viewport's* larger dimension
cm	centimeters
mm	millimeters
in	inches (1in = 96px = 2.54cm)
px	pixels (1px = 1/96th of 1in)
pt	points (1pt = 1/72 of 1in)
pc	picas (1pc = 12 pt)
s	seconds (used for animations and transitions)

Unit	Description
ms	milliseconds (used for animations and transitions)
ex	Relative to the x-height of the current font
ch	Based on the width of the zero (0) character
fr	fractional unit (used for CSS Grid Layout)

Remarks

- A whitespace cannot appear between the number and the unit. However, if the value is 0, the unit can be omitted.
- For some CSS properties, negative lengths are allowed.

Examples

Font size with rem

CSS3 introduces a few new units, including the `rem` unit, which stands for "root em". Let's look at how `rem` works.

First, let's look at the differences between `em` and `rem`.

- **em**: Relative to the font size of the parent. This causes the compounding issue
- **rem**: Relative to the font size of the root or `<html>` element. This means it's possible to declare a single font size for the html element and define all `rem` units to be a percentage of that.

The main issue with using `rem` for font sizing is that the values are somewhat difficult to use. Here is an example of some common font sizes expressed in `rem` units, assuming that the base size is 16px :

- 10px = 0.625rem
- 12px = 0.75rem
- 14px = 0.875rem
- 16px = 1rem (base)
- 18px = 1.125rem
- 20px = 1.25rem
- 24px = 1.5rem
- 30px = 1.875rem
- 32px = 2rem

CODE:

3

```

html {
  font-size: 16px;
}

h1 {
  font-size: 2rem;          /* 32px */
}

p {
  font-size: 1rem;          /* 16px */
}

li {
  font-size: 1.5em;         /* 24px */
}

```

Creating scalable elements using rems and ems

3

You can use `rem` defined by the `font-size` of your `html` tag to style elements by setting their `font-size` to a value of `rem` and use `em` inside the element to create elements that scale with your global `font-size`.

HTML:

```





```

Relevant CSS:

```

html {
  font-size: 16px;
}

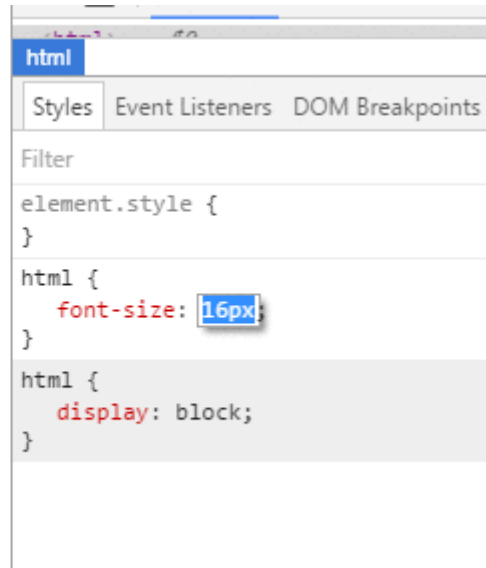
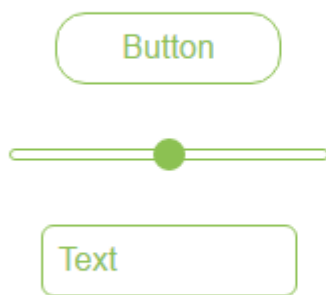
input[type="button"] {
  font-size: 1rem;
  padding: 0.5em 2em;
}

input[type="range"] {
  font-size: 1rem;
  width: 10em;
}

input[type="text"] {
  font-size: 1rem;
  padding: 0.5em;
}

```

Possible Result:



vh and vw

CSS3 introduced two units for representing size.

- **vh**, which stands for `viewport height` is relative to 1% of the viewport height
- **vw**, which stands for `viewport width` is relative to 1% of the viewport width

3

```
div {
  width: 20vw;
  height: 20vh;
}
```

Above, the size for the div takes up 20% of the width and height of the viewport

vmin and vmax

- **vmin**: Relative to 1 percent of the viewport's smaller dimension
- **vmax**: Relative to 1 percent of the viewport's larger dimension

In other words, 1 `vmin` is equal to the smaller of 1 `vh` and 1 `vw`

1 `vmax` is equal to the larger of 1 `vh` and 1 `vw`

Note: `vmax` is **not supported** in:

- any version of Internet Explorer
- Safari before version 6.1

using percent %

One of the useful unit when creating a responsive application.

Its size depends on its parent container.

Equation:

$(\text{Parent Container's width}) * (\text{Percentage}(\%)) = \text{Output}$

For Example:

Parent has **100px** width while the *Child* has **50%**.

On the output, the *Child's* width will be half(50%) of the *Parent's*, which is **50px**.

HTML

```
<div class="parent">
  PARENT
  <div class="child">
    CHILD
  </div>
</div>
```

CSS

```
<style>

*{
  color: #CCC;
}

.parent{
  background-color: blue;
  width: 100px;
}

.child{
  background-color: green;
  width: 50%;
}

</style>
```

OUTPUT



Read Length Units online: <https://riptutorial.com/css/topic/864/length-units>

Chapter 34: List Styles

Syntax

- list-style: list-style-type | list-style-position | list-style-image | initial | inherit;

Parameters

Value	Description
list-style-type	the type of list-item marker.
list-style-position	specifies where to place the marker
list-style-image	specifies the type of list-item marker
initial	sets this property to its default value
inherit	inherits this property from its parent element

Remarks

Although the `list-style-type` is actually a property that applies only to list items (normally `<li>`), it is often specified for the list tag (`<ol>` or `<ul>`). In this case, the list items inherit the property.

Examples

Type of Bullet or Numbering

Specific for `<li>` tags within an unordered list (`<ul>`):

```
list-style: disc;           /* A filled circle (default) */
list-style: circle;        /* A hollow circle */
list-style: square;        /* A filled square */
list-style: '-';           /* any string */
```

Specific for `<li>` tags within an ordered list (`<ol>`):

```
list-style: decimal;        /* Decimal numbers beginning with 1 (default) */
list-style: decimal-leading-zero; /* Decimal numbers padded by initial zeros (01, 02, 03, ... 10) */
list-style: lower-roman;    /* Lowercase roman numerals (i., ii., iii., iv., ...) */
list-style: upper-roman;    /* Uppercase roman numerals (I., II., III., IV., ...) */
list-style-type: lower-greek; /* Lowercase greek letters (α., β., γ., δ., ...) */
list-style-type: lower-alpha; /* Lowercase letters (a., b., c., d., ...) */
list-style-type: lower-latin; /* Lowercase letters (a., b., c., d., ...) */
```

```
list-style-type: upper-alpha; /* Uppercase letters (A., B., C., D., ...) */
list-style-type: upper-latin; /* Uppercase letters (A., B., C., D., ...) */
```

Non-specific:

```
list-style: none; /* No visible list marker */
list-style: inherit; /* Inherits from parent */
```

Bullet Position

A list consists of `<li>` elements inside a containing element (`<ul>` or `<ol>`). Both the list items and the container can have margins and paddings which influence the exact position of the list item content in the document. The default values for the margin and padding may be different for each browser. In order to get the same layout cross-browser, these need to be set specifically.

Each list item gets a 'marker box', which contains the bullet marker. This box can either be placed inside or outside of the list item box.

```
list-style-position: inside;
```

places the bullet within the `<li>` element, pushing the content to the right as needed.

```
list-style-position: outside;
```

places the bullet left of the `<li>` element. If there is not enough space in the padding of the containing element, the marker box will extend to the left even if it would fall off the page.

Showing the result of `inside` and `outside` positioning: [jsfiddle](#)

Removing Bullets / Numbers

Sometimes, a list should just not display any bullet points or numbers. In that case, remember to specify margin and padding.

```
<ul>
  <li>first item</li>
  <li>second item</li>
</ul>
```

CSS

```
ul {
  list-style-type: none;
}
li {
  margin: 0;
  padding: 0;
}
```

Read List Styles online: <https://riptutorial.com/css/topic/4215/list-styles>

Chapter 35: Margins

Syntax

- `margin: <top & right & bottom & left>;`
- `margin: <top>, <left & right>, <bottom>;`
- `margin: <top & bottom>, <left & right>;`
- `margin: <top>, <right>, <bottom>, <left>;`
- `margin-top: <top>;`
- `margin-right: <right>;`
- `margin-bottom: <bottom>;`
- `margin-left: <left>;`

Parameters

Parameter	Details
0	set margin to none
auto	used for centering, by evenly setting values on each side
units (e.g. px)	see parameter section in Units for a list of valid units
inherit	inherit margin value from parent element
initial	restore to initial value

Remarks

More on "Collapsing Margins": [here](#).

Examples

Apply Margin on a Given Side

Direction-Specific Properties

CSS allows you to specify a given side to apply margin to. The four properties provided for this purpose are:

- `margin-left`
- `margin-right`
- `margin-top`

- `margin-bottom`

The following code would apply a margin of 30 pixels to the left side of the selected div. [View Result](#)

HTML

```
<div id="myDiv"></div>
```

CSS

```
#myDiv {  
    margin-left: 30px;  
    height: 40px;  
    width: 40px;  
    background-color: red;  
}
```

Parameter	Details
<code>margin-left</code>	The direction in which the margin should be applied.
<code>30px</code>	The width of the margin.

Specifying Direction Using Shorthand Property

The standard `margin` property can be expanded to specify differing widths to each side of the selected elements. The syntax for doing this is as follows:

```
margin: <top> <right> <bottom> <left>;
```

The following example applies a zero-width margin to the top of the div, a 10px margin to the right side, a 50px margin to the left side, and a 100px margin to the left side. [View Result](#)

HTML

```
<div id="myDiv"></div>
```

CSS

```
#myDiv {  
    margin: 0 10px 50px 100px;  
    height: 40px;  
    width: 40px;  
    background-color: red;  
}
```

Margin Collapsing

When two margins are touching each other vertically, they are collapsed. When two margins touch horizontally, they do not collapse.

Example of adjacent vertical margins:

Consider the following styles and markup:

```
div{  
    margin: 10px;  
}
```

```
<div>  
    some content  
</div>  
<div>  
    some more content  
</div>
```

They will be 10px apart since vertical margins collapse over one and other. (The spacing will not be the sum of two margins.)

Example of adjacent horizontal margins:

Consider the following styles and markup:

```
span{  
    margin: 10px;  
}
```

```
<span>some</span><span>content</span>
```

They will be 20px apart since horizontal margins don't collapse over one and other. (The spacing will be the sum of two margins.)

Overlapping with different sizes

```
.top{  
    margin: 10px;  
}  
.bottom{  
    margin: 15px;  
}
```

```
<div class="top">  
    some content  
</div>  
<div class="bottom">  
    some more content  
</div>
```


These elements will be spaced 15px apart vertically. The margins overlap as much as they can, but the larger margin will determine the spacing between the elements.

Overlapping margin gotcha

```
.outer-top{
  margin: 10px;
}
.inner-top{
  margin: 15px;
}
.outer-bottom{
  margin: 20px;
}
.inner-bottom{
  margin: 25px;
}
```

```
<div class="outer-top">
  <div class="inner-top">
    some content
  </div>
</div>
<div class="outer-bottom">
  <div class="inner-bottom">
    some more content
  </div>
</div>
```

What will be the spacing between the two texts? (hover to see answer)

The spacing will be 25px. Since all four margins are touching each other, they will collapse, thus using the largest margin of the four.

Now, what about if we add some borders to the markup above.

```
div{
  border: 1px solid red;
}
```

What will be the spacing between the two texts? (hover to see answer)

The spacing will be 59px! Now only the margins of .outer-top and .outer-bottom touch each other, and are the only collapsed margins. The remaining margins are separated by the borders. So we have 1px + 10px + 1px + 15px + 20px + 1px + 25px + 1px. (The 1px's are the borders...)

Collapsing Margins Between Parent and Child Elements:

HTML:

```
<h1>Title</h1>
<div>
  <p>Paragraph</p>
```

```
</div>
```

CSS

```
h1 {
  margin: 0;
  background: #cff;
}
div {
  margin: 50px 0 0 0;
  background: #cfc;
}
p {
  margin: 25px 0 0 0;
  background: #cf9;
}
```

In the example above, only the largest margin applies. You may have expected that the paragraph would be located 60px from the h1 (since the div element has a margin-top of 40px and the p has a 20px margin-top). This does not happen because the margins collapse together to form one margin.

Horizontally center elements on a page using margin

As long as the element is a **block**, and it has an **explicitly set width value**, margins can be used to center block elements on a page horizontally.

We add a width value that is lower than the width of the window and the auto property of margin then distributes the remaining space to the left and the right:

```
#myDiv {
  width:80%;
  margin:0 auto;
}
```

In the example above we use the shorthand `margin` declaration to first set `0` to the top and bottom margin values (although this could be any value) and then we use `auto` to let the browser allocate the space automatically to the left and right margin values.

In the example above, the `#myDiv` element is set to 80% width which leaves use 20% leftover. The browser distributes this value to the remaining sides so:

$$(100\% - 80\%) / 2 = 10\%$$

Margin property simplification

```
p {
  margin:1px;                      /* 1px margin in all directions */

  /*equals to:*/

  margin:1px 1px;
```

```

/*equals to:*/

margin:1px 1px 1px;

/*equals to:*/

margin:1px 1px 1px 1px;
}

```

Another exapmle:

```

p{
  margin:10px 15px;          /* 10px margin-top & bottom And 15px margin-right & left*/

  /*equals to:*/

  margin:10px 15px 10px 15px;

  /*equals to:*/

  margin:10px 15px 10px;
  /* margin left will be calculated from the margin right value (=15px) */
}

```

Negative margins

Margin is one of a few CSS properties that can be set to negative values. This property can be used to **overlap elements without absolute positioning**.

```

div{
  display: inline;
}

#over{
  margin-left: -20px;
}

<div>Base div</div>
<div id="over">Overlapping div</div>

```

Example 1:

It is obvious to assume that the percentage value of margin to `margin-left` and `margin-right` would be relative to its parent element.

```

.parent {
  width : 500px;
  height: 300px;
}

.child {
  width : 100px;
  height: 100px;
  margin-left: 10%; /* (parentWidth * 10/100) => 50px */
}

```

```
}
```

But that is not the case, when comes to `margin-top` and `margin-bottom`. Both these properties, in percentages, aren't relative to the height of the parent container but to the **width** of the parent container.

So,

```
.parent {  
  width : 500px;  
  height: 300px;  
}  
  
.child {  
  width : 100px;  
  height: 100px;  
  margin-left: 10%; /* (parentWidth * 10/100) => 50px */  
  margin-top: 20%; /* (parentWidth * 20/100) => 100px */  
}
```

Read Margins online: <https://riptutorial.com/css/topic/305/margins>

Chapter 36: Media Queries

Syntax

- `@media [not|only] mediatype and (media feature) { /* CSS rules to apply */ }`

Parameters

Parameter	Details
<code>mediatype</code>	(Optional) This is the type of media. Could be anything in the range of <code>all</code> to <code>screen</code> .
<code>not</code>	(Optional) Doesn't apply the CSS for this particular media type and applies for everything else.
<code>media feature</code>	Logic to identify use case for CSS. Options outlined below.
Media Feature	Details
<code>aspect-ratio</code>	Describes the aspect ratio of the targeted display area of the output device.
<code>color</code>	Indicates the number of bits per color component of the output device. If the device is not a color device, this value is zero.
<code>color-index</code>	Indicates the number of entries in the color look-up table for the output device.
<code>grid</code>	Determines whether the output device is a grid device or a bitmap device.
<code>height</code>	The height media feature describes the height of the output device's rendering surface.
<code>max-width</code>	CSS will not apply on a screen width wider than specified.
<code>min-width</code>	CSS will not apply on a screen width narrower than specified.
<code>max-height</code>	CSS will not apply on a screen height taller than specified.
<code>min-height</code>	CSS will not apply on a screen height shorter than specified.
<code>monochrome</code>	Indicates the number of bits per pixel on a monochrome (greyscale) device.
<code>orientation</code>	CSS will only display if device is using specified orientation. See remarks for more details.

Parameter	Details
resolution	Indicates the resolution (pixel density) of the output device.
scan	Describes the scanning process of television output devices.
width	The width media feature describes the width of the rendering surface of the output device (such as the width of the document window, or the width of the page box on a printer).
Deprecated Features	Details
device-aspect-ratio	Deprecated CSS will only display on devices whose height/width ratio matches the specified ratio. This is a deprecated feature and is not guaranteed to work.
max-device-width	Deprecated Same as <code>max-width</code> but measures the physical screen width, rather than the display width of the browser.
min-device-width	Deprecated Same as <code>min-width</code> but measures the physical screen width, rather than the display width of the browser.
max-device-height	Deprecated Same as <code>max-height</code> but measures the physical screen width, rather than the display width of the browser.
min-device-height	Deprecated Same as <code>min-height</code> but measures the physical screen width, rather than the display width of the browser.

Remarks

Media queries are supported in all modern browsers, including Chrome, Firefox, Opera, and Internet Explorer 9 and up.

It is important to note that the `orientation` media feature is not limited to mobile devices. It is based on the width and height of the viewport (not window or devices).

Landscape Mode is when the viewport width is larger than viewport height.

Portrait Mode is when the viewport height is larger than viewport width.

This usually translates to a desktop monitor being in landscape mode, but can it sometimes be portrait.

In most cases mobile devices will report their resolution and not their real pixel size which can differ due to pixel density. To force them to report their real pixel size add the following inside your head tag:

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

Examples

Basic Example

```
@media screen and (min-width: 720px) {  
  body {  
    background-color: skyblue;  
  }  
}
```

The above media query specifies two conditions:

1. The page must be viewed on a normal screen (not a printed page, projector, etc).
2. The width of the user's view port must be at least 720 pixels.

If these conditions are met, the styles inside the media query will be active, and the background color of the page will be sky blue.

Media queries are applied dynamically. If on page load the conditions specified in the media query are met, the CSS will be applied, but will be immediately disabled should the conditions cease to be met. Conversely, if the conditions are initially not met, the CSS will not be applied until the specified conditions are met.

In our example, if the user's view port width is initially greater than 720 pixels, but the user shrinks the browser's width, the background color will cease to be sky blue as soon as the user has resized the view port to less than 720 pixels in width.

Use on link tag

```
<link rel="stylesheet" media="min-width: 600px" href="example.css" />
```

This stylesheet is still downloaded but is applied only on devices with screen width larger than 600px.

mediatype

Media queries have an optional `mediatype` parameter. This parameter is placed directly after the `@media` declaration (`@media mediatype`), for example:

```
@media print {  
  html {  
    background-color: white;  
  }  
}
```

The above CSS code will give the DOM `HTML` element a white background color when being

printed.

The `mediatype` parameter has an optional `not` or `only` prefix that will apply the styles to everything except the specified mediatype *or* only the specified media type, respectively. For example, the following code example will apply the style to every media type except `print`.

```
@media not print {  
  html {  
    background-color: green;  
  }  
}
```

And the same way, for just showing it only on the screen, this can be used:

```
@media only screen {  
  .fadeInEffects {  
    display: block;  
  }  
}
```

The list of `mediatype` can be understood better with the following table:

Media Type	Description
all	Apply to all devices
screen	Default computers
print	Printers in general. Used to style print-versions of websites
handheld	PDA's, cellphones and hand-held devices with a small screen
projection	For projected presentation, for example projectors
aural	Speech Systems
braille	Braille tactile devices
embossed	Paged braille printers
tv	Television-type devices
tty	Devices with a fixed-pitch character grid. Terminals, portables.

Using Media Queries to Target Different Screen Sizes

Often times, responsive web design involves media queries, which are CSS blocks that are only executed if a condition is satisfied. This is useful for responsive web design because you can use media queries to specify different CSS styles for the mobile version of your website versus the desktop version.


```

@media only screen and (min-width: 300px) and (max-width: 767px) {
    .site-title {
        font-size: 80%;
    }

    /* Styles in this block are only applied if the screen size is atleast 300px wide, but no
more than 767px */
}

@media only screen and (min-width: 768px) and (max-width: 1023px) {
    .site-title {
        font-size: 90%;
    }

    /* Styles in this block are only applied if the screen size is atleast 768px wide, but no
more than 1023px */
}

@media only screen and (min-width: 1024px) {
    .site-title {
        font-size: 120%;
    }

    /* Styles in this block are only applied if the screen size is over 1024px wide. */
}

```

Width vs Viewport

When we are using "width" with media queries it is important to set the meta tag correctly. Basic meta tag looks like this and it needs to be put inside the `<head>` tag.

```
<meta name="viewport" content="width=device-width,initial-scale=1">
```

Why this is important?

Based on MDN's definition "width" is

The width media feature describes the width of the rendering surface of the output device (such as the width of the document window, or the width of the page box on a printer).

What does that mean?

View-portal is the width of the device itself. If your screen resolution says the resolution is 1280 x 720, your view-portal width is "1280px".

More often many devices allocate different pixel amount to display one pixel. For an example iPhone 6 Plus has 1242 x 2208 resolution. But the actual view-portal-width and view-portal-height is 414 x 736. That means 3 pixels are used to create 1 pixel.

But if you did not set the `meta` tag correctly it will try to show your webpage with its native resolution which results in a zoomed out view (smaller texts and images).

Media Queries for Retina and Non Retina Screens

Although this works only for WebKit based browsers, this is helpful:

```
/* ----- Non-Retina Screens ----- */
@media screen
  and (min-width: 1200px)
  and (max-width: 1600px)
  and (-webkit-min-device-pixel-ratio: 1) {

/* ----- Retina Screens ----- */
@media screen
  and (min-width: 1200px)
  and (max-width: 1600px)
  and (-webkit-min-device-pixel-ratio: 2)
  and (min-resolution: 192dpi) {

}
```

Background Information

There are two types of pixels in the display. One is the logical pixels and the other is the physical pixels. Mostly, the physical pixels always stay the same, because it is the same for all the display devices. The logical pixels change based on the resolution of the devices to display higher quality pixels. The device pixel ratio is the ratio between physical pixels and logical pixels. For instance, the MacBook Pro Retina, iPhone 4 and above report a device pixel ratio of 2, because the physical linear resolution is double the logical resolution.

The reason why this works only with WebKit based browsers is because of:

- The vendor prefix `-webkit-` before the rule.
- This hasn't been implemented in engines other than WebKit and Blink.

Terminology and Structure

Media queries allow one to apply CSS rules based on the type of device / media (e.g. screen, print or handheld) called **media type**, additional aspects of the device are described with **media features** such as the availability of color or viewport dimensions.

General Structure of a Media Query

```
@media [...] {
  /* One or more CSS rules to apply when the query is satisfied */
}
```

A Media Query containing a Media Type

```
@media print {
```

```
/* One or more CSS rules to apply when the query is satisfied */  
}
```

A Media Query containing a Media Type and a Media Feature

```
@media screen and (max-width: 600px) {  
    /* One or more CSS rules to apply when the query is satisfied */  
}
```

A Media Query containing a Media Feature (and an implicit Media Type of "all")

```
@media (orientation: portrait) {  
    /* One or more CSS rules to apply when the query is satisfied */  
}
```

Media queries and IE8

[Media queries](#) are not supported at all in IE8 and below.

A Javascript based workaround

To add support for IE8, you could use one of several JS solutions. For example, [Respond](#) can be added to add media query support for IE8 only with the following code :

```
<!--[if lt IE 9]>  
<script  
    src="respond.min.js">  
</script>  
<![endif]-->
```

[CSS Mediaqueries](#) is another library that does the same thing. The code for adding that library to your HTML would be identical :

```
<!--[if lt IE 9]>  
<script  
    src="css3-mediaqueries.js">  
</script>  
<![endif]-->
```

The alternative

If you don't like a JS based solution, you should also consider adding an IE<9 only stylesheet where you adjust your styling specific to IE<9. For that, you should add the following HTML to your code:

```
<!--[if lt IE 9]>
<link rel="stylesheet" type="text/css" media="all" href="style-ielt9.css"/>
<![endif]-->
```

Note :

Technically it's one more alternative: using **CSS hacks** to target IE<9. It has the same impact as an IE<9 only stylesheet, but you don't need a separate stylesheet for that. I do not recommend this option, though, as they produce invalid CSS code (which is but one of several reasons why the use of CSS hacks is generally frowned upon today).

Read Media Queries online: <https://riptutorial.com/css/topic/317/media-queries>

Chapter 37: Multiple columns

Introduction

CSS allows to define that element contents wrap into multiple columns with gaps and rules between them.

Remarks

[CSS Multi-column Layout Module Level 1](#) is, as of 12 April 2011, a W3C Candidate Recommendation. Since then, a few [smaller changes were made](#). It is considered to be in the [Stable stage](#).

As of 3 July 2017, Microsoft's Internet Explorer 10 and 11 and Edge browsers only support an older version of the specification using a vendor prefix.

Examples

Basic example

Consider the following HTML markup:

```
<section>
  <p>Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum.</p>
  <p>Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.</p>
  <p>Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.</p>
</section>
```

With the following CSS applied the content is split into three columns separated by a gray column rule of two pixels.

```
section {
  columns: 3;
  column-gap: 40px;
  column-rule: 2px solid gray;
}
```

See a [live sample of this on JSFiddle](#).

Create Multiple Columns

```
<div class="content">
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam nonummy nibh
euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim
ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl
ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in
hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu
feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui
blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla
facilisi. Nam liber tempor cum soluta nobis eleifend option congue nihil
imperdiet doming id quod mazim placerat facer possim assum.
</div>
```

Css

```
.content {
-webkit-column-count: 3; /* Chrome, Safari, Opera */
-moz-column-count: 3; /* Firefox */
column-count: 3;
}
```

Read Multiple columns online: <https://riptutorial.com/css/topic/10688/multiple-columns>

Chapter 38: Normalizing Browser Styles

Introduction

Every browser has a default set of CSS styles that it uses for rendering elements. These default styles may not be consistent across browsers because: the language specifications are unclear so base styles are up for interpretation, browsers may not follow specifications that are given, or browsers may not have default styles for newer HTML elements. As a result, people may want to normalize default styles across as many browsers as possible.

Remarks

Meyer's Reset, while effective, makes the same changes to nearly every commonly used element. This has the result of polluting web browser inspector windows with the same applied styles over and over again, and creates more work for the browser (more rules to apply to more elements). The Normalize technique, on the other hand, is much more focused and less of a broad-brush technique. This simplifies the work on the part of the browser, and results in less clutter in the browser inspection tools.

Examples

`normalize.css`

Browsers have a default set of CSS styles they use for rendering elements. Some of these styles can even be customised using the browser's settings to change default font face and size definitions, for example. The styles contain the definition of which elements are supposed to be block-level or inline, among other things.

Because these default styles are given some leeway by the language specifications and because browsers may not follow the specs properly they can differ from browser to browser.

This is where [normalize.css](https://necolas.github.io/normalize.css/) comes into play. It overrides the most common inconsistencies and fixes known bugs.

What does it do

- Preserves useful defaults, unlike many CSS resets.
- Normalizes styles for a wide range of elements.
- Corrects bugs and common browser inconsistencies.
- Improves usability with subtle modifications.
- Explains what code does using detailed comments.

So, by including `normalize.css` in your project your design will look more alike and consistent across different browsers.

Difference to reset.css

You may have heard of `reset.css`. What's the difference between the two?

While `normalize.css` provides consistency by setting different properties to unified defaults, `reset.css` achieves consistency by **removing** all basic styling that a browser may apply. While this might sound like a good idea at first, this actually means you have to write **all** rules yourself, which goes against having a solid standard.

Approaches and Examples

CSS resets take separate approaches to browser defaults. Eric Meyer's Reset CSS has been around for a while. His approach nullifies many of the browser elements that have been known to cause problems right off the back. The following is from his version (v2.0 | 20110126) CSS Reset.

```
html, body, div, span, applet, object, iframe,
h1, h2, h3, h4, h5, h6, p, blockquote, pre,
a, abbr, acronym, address, big, cite, code,
del, dfn, em, img, ins, kbd, q, s, samp,
small, strike, strong, sub, sup, tt, var,
b, u, i, center,
dl, dt, dd, ol, ul, li,
fieldset, form, label, legend,
table, caption, tbody, tfoot, thead, tr, th, td,
article, aside, canvas, details, embed,
figure, figcaption, footer, header, hgroup,
menu, nav, output, ruby, section, summary,
time, mark, audio, video {
    margin: 0;
    padding: 0;
    border: 0;
    font-size: 100%;
    font: inherit;
    vertical-align: baseline;
}
```

Eric Meyer's Reset CSS

Normalize CSS on the other and deals with many of these separately. The following is a sample from the version (v4.2.0) of the code.

```
/**
 * 1. Change the default font family in all browsers (opinionated).
 * 2. Correct the line height in all browsers.
 * 3. Prevent adjustments of font size after orientation changes in IE and iOS.
 */

/* Document
===== */

html {
    font-family: sans-serif; /* 1 */
    line-height: 1.15; /* 2 */
    -ms-text-size-adjust: 100%; /* 3 */
```



```

    -webkit-text-size-adjust: 100%; /* 3 */
}

/* Sections
   ===== */

/**
 * Remove the margin in all browsers (opinionated).
 */

body {
    margin: 0;
}

/**
 * Add the correct display in IE 9-.
 */

article,
aside,
footer,
header,
nav,
section {
    display: block;
}

/**
 * Correct the font size and margin on `h1` elements within `section` and
 * `article` contexts in Chrome, Firefox, and Safari.
 */

h1 {
    font-size: 2em;
    margin: 0.67em 0;
}

```

Normalize CSS

Read Normalizing Browser Styles online: <https://riptutorial.com/css/topic/1211/normalizing-browser-styles>

Chapter 39: Object Fit and Placement

Remarks

The properties `object-fit` and `object-position` are not supported by Internet Explorer.

Examples

`object-fit`

The **`object-fit`** property will defines how an element will fit into a box with an established height and width. Usually applied to an image or video, Object-fit accepts the following five values:

FILL

```
object-fit:fill;
```

original image



`object-fit: fill;`



Fill stretches the image to fit the content box without regard to the image's original aspect ratio.

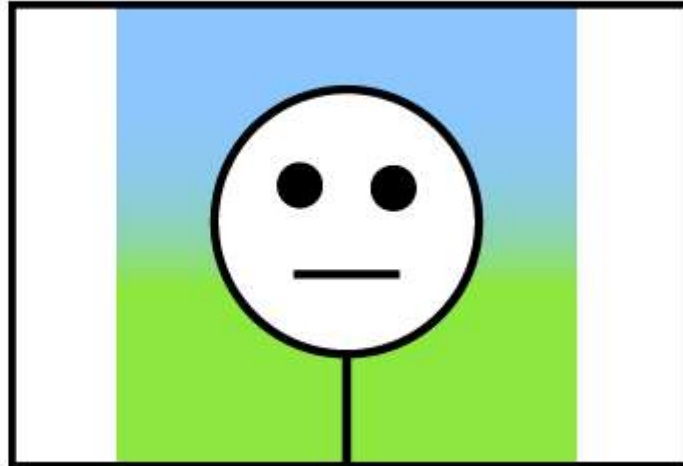
CONTAIN

```
object-fit:contain;
```

original image



object-fit: contain;

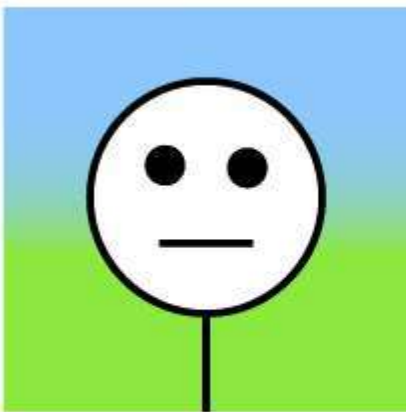


Contain fits the image in the box's height or width while maintaining the image's aspect ratio.

COVER

```
object-fit: cover;
```

original image



object-fit: cover;



Cover fills the entire box with the image. The image aspect ratio is preserved, but the image is cropped to the dimensions of the box.

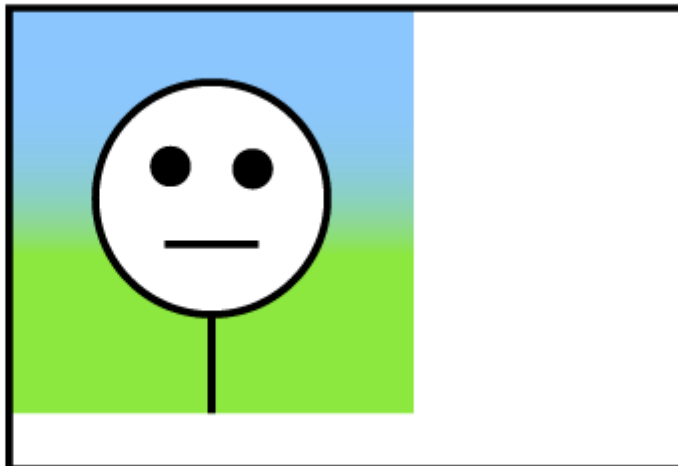
NONE

```
object-fit: none;
```

original image



object-fit: none;



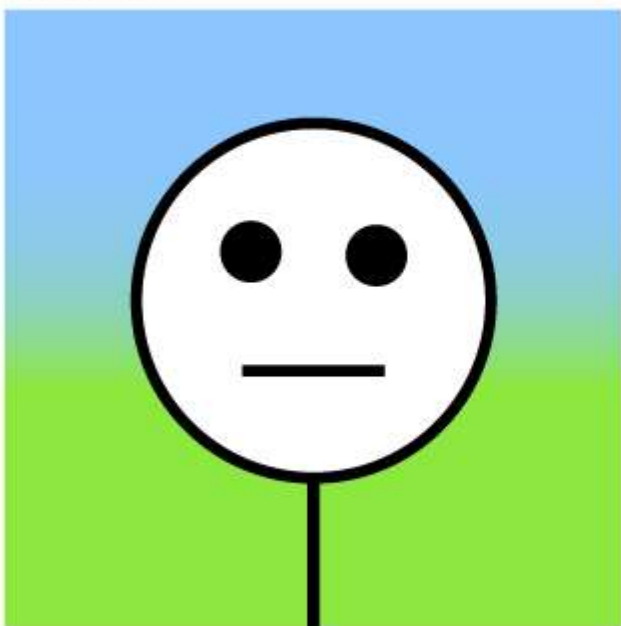
None ignores the size of the box and is not resized.

SCALE-DOWN

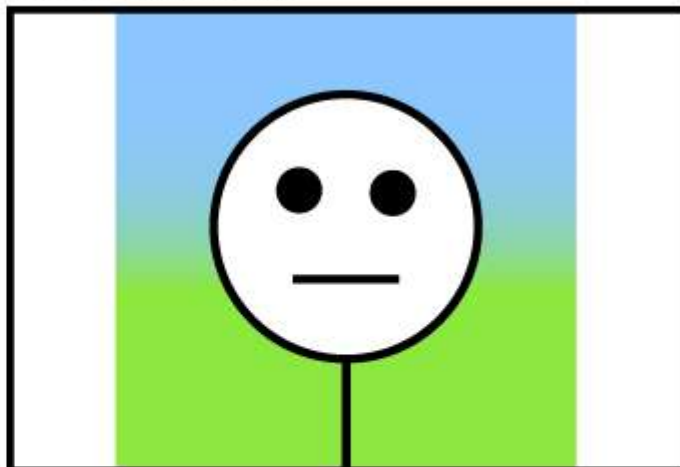
```
object-fit: scale-down;
```

Scale-down either sizes the object as `none` or as `contain`. It displays whichever option results in a smaller image size.

original image



object-fit: scale-down;



Read Object Fit and Placement online: <https://riptutorial.com/css/topic/5520/object-fit-and-placement>

Chapter 40: Opacity

Syntax

- opacity: number (* strictly between 0 and 1) | inherit | initial | unset;

Remarks

If you do not want apply opacity, you can use this instead:

background: rgba(255, 255, 255, 0.6);

Resources:

- MDN: <https://developer.mozilla.org/en/docs/Web/CSS/opacity>;
- W3C Transparency: the 'opacity' property: <https://www.w3.org/TR/css3-color/#transparency>
- Browser support: <http://caniuse.com/#feat=css-opacity>

Examples

Opacity Property

An element's opacity can be set using the `opacity` property. Values can be anywhere from 0.0 (transparent) to 1.0 (opaque).

Example Usage

```
<div style="opacity:0.8;">  
  This is a partially transparent element  
</div>
```

Property Value	Transparency
opacity: 1.0;	Opaque
opacity: 0.75;	25% transparent (75% Opaque)
opacity: 0.5;	50% transparent (50% Opaque)
opacity: 0.25;	75% transparent (25% Opaque)
opacity: 0.0;	Transparent

IE Compatibility for `opacity`

To use `opacity` in all versions of IE, the order is:

```
.transparent-element {  
  /* for IE 8 & 9 */  
  -ms-filter:"progid:DXImageTransform.Microsoft.Alpha(Opacity=60)"; // IE8  
  /* works in IE 8 & 9 too, but also 5, 6, 7 */  
  filter: alpha(opacity=60); // IE 5-7  
  /* Modern Browsers */  
  opacity: 0.6;  
}
```

Read Opacity online: <https://riptutorial.com/css/topic/2864/opacity>

Chapter 41: Outlines

Syntax

- `outline`: `outline-color outline-style outline-width` | `initial` | `inherit`;
- `outline-width`: `medium` | `thin` | `thick` | `length` | `initial` | `inherit`;
- `outline-style`: `none` | `hidden` | `dotted` | `dashed` | `solid` | `double` | `groove` | `ridge` | `inset` | `outset` | `initial` | `inherit`;

Parameters

Parameter	Details
dotted	dotted outline
dashed	dashed outline
solid	solid outline
double	double outline
groove	3D grooved outline, depends on the <code>outline-color</code> value
ridge	3D ridged outline, depends on the <code>outline-color</code> value
inset	3D inset outline, depends on the <code>outline-color</code> value
outset	3D outset outline, depends on the <code>outline-color</code> value
none	no outline
hidden	hidden outline

Remarks

`outline` is now described in [Basic UI](#), a CSS Module Level 3 (it was already described in REC CSS2.1)

Outline property is defined by default in browsers for focusable elements in `:focus` state. It shouldn't be removed, see <http://outlinenone.com> which states:

What does the outline property do?

It provides visual feedback for links that have "focus" when navigating a web document using the TAB key (or equivalent). This is especially useful for folks who can't use a

mouse or have a visual impairment. If you remove the outline you are making your site inaccessible for these people. (...)

Interesting related examples on Stack Overflow:

- [How to remove the border highlight on an input text element](#)
- [How to remove Firefox's dotted outline on BUTTONS as well as links?](#)

Examples

Overview

Outline is a line that goes around the element, outside of the border. In contrast to `border`, outlines do not take any space in the box model. So adding an outline to an element does not affect the position of the element or other elements.

In addition, outlines can be non-rectangular in some browsers. This can happen if `outline` is applied on a `span` element that has text with different `font-size` properties inside it. Unlike borders, outlines *cannot* have rounded corners.

The essential parts of `outline` are `outline-color`, `outline-style` and `outline-width`.

The definition of an outline is equivalent to the definition of a border:

An outline is a line around an element. It is displayed around the margin of the element. However, it is different from the border property.

```
outline: 1px solid black;
```

outline-style

The `outline-style` property is used to set the style of the outline of an element.

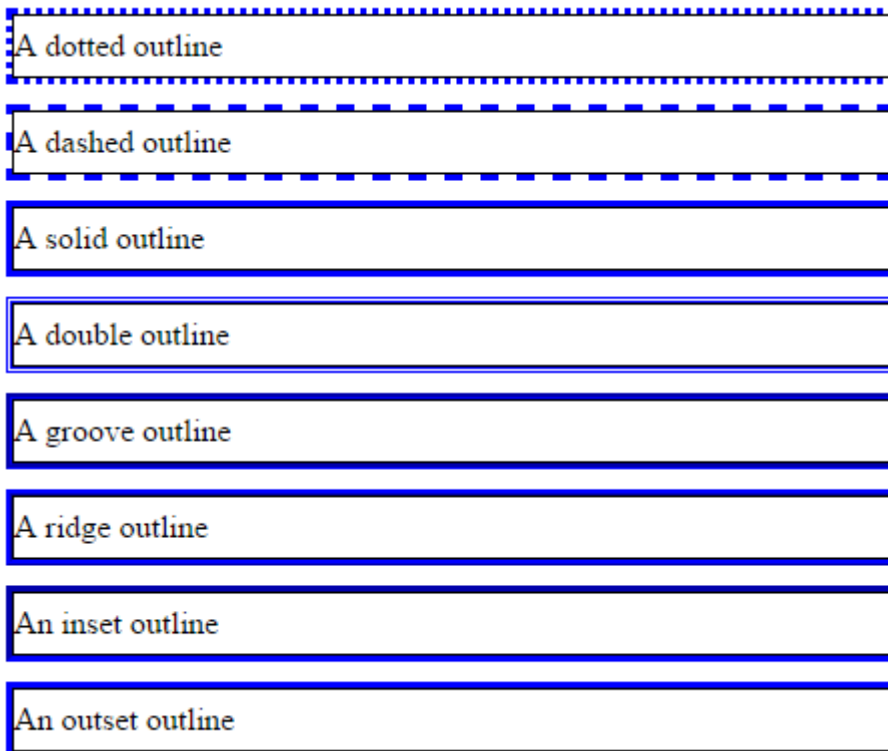
```
p {
  border: 1px solid black;
  outline-color:blue;
  line-height:30px;
}
.p1{
  outline-style: dotted;
}
.p2{
  outline-style: dashed;
}
.p3{
  outline-style: solid;
}
.p4{
  outline-style: double;
}
.p5{
  outline-style: groove;
```



```
}  
.p6{  
  outline-style: ridge;  
}  
.p7{  
  outline-style: inset;  
}  
.p8{  
  outline-style: outset;  
}
```

HTML

```
<p class="p1">A dotted outline</p>  
<p class="p2">A dashed outline</p>  
<p class="p3">A solid outline</p>  
<p class="p4">A double outline</p>  
<p class="p5">A groove outline</p>  
<p class="p6">A ridge outline</p>  
<p class="p7">An inset outline</p>  
<p class="p8">An outset outline</p>
```



Read Outlines online: <https://riptutorial.com/css/topic/4258/outlines>

Chapter 42: Overflow

Syntax

- `overflow: visible | hidden | scroll | auto | initial | inherit;`

Parameters

Overflow Value	Details
<code>visible</code>	Shows all overflowing content outside the element
<code>scroll</code>	Hides the overflowing content and adds a scroll bar
<code>hidden</code>	Hides the overflowing content, both scroll bars disappear and the page becomes fixed
<code>auto</code>	Same as <code>scroll</code> if content overflows, but doesn't add scroll bar if content fits
<code>inherit</code>	Inherit's the parent element's value for this property

Remarks

The `overflow` property specifies whether to clip content, render scrollbars, or stretch a container to display content when it overflows its block level container.

When an element is too small to display it's contents, what happens? By default, the content will just overflow and display outside the element. That makes your work look bad. You want your work to look good, so you set the `overflow` property to handle the overflowing content in a desirable way.

Values for the `overflow` property are identical to those for the `overflow-x` and `overflow-y` properties, except that they apply along each axis

The `overflow-wrap` property has also been known as the `word-wrap` property.

Important note: Using the `overflow` property with a value different to `visible` will create a new block formatting context.

Examples

overflow: scroll

HTML

```
<div>
  This div is too small to display its contents to display the effects of the overflow
  property.
</div>
```

CSS

```
div {
  width:100px;
  height:100px;
  overflow:scroll;
}
```

Result



The content above is clipped in a 100px by 100px box, with scrolling available to view overflowing content.

Most desktop browsers will display both horizontal and vertical scrollbars, whether or not any content is clipped. This can avoid problems with scrollbars appearing and disappearing in a dynamic environment. Printers may print overflowing content.

overflow-wrap

`overflow-wrap` tells a browser that it can break a line of text inside a targeted element onto multiple lines in an otherwise unbreakable place. Helpful in preventing an long string of text causing layout problems due to overflowing it's container.

CSS

```
div {
  width:100px;
  outline: 1px dashed #bbb;
}

#div1 {
  overflow-wrap:normal;
}

#div2 {
  overflow-wrap:break-word;
}
```

HTML

```
<div id="div1">
  <strong>#div1</strong>: Small words are displayed normally, but a long word like <span
style="red;">supercalifragilisticexpialidocious</span> is too long so it will overflow past
the edge of the line-break
</div>

<div id="div2">
  <strong>#div2</strong>: Small words are displayed normally, but a long word like <span
style="red;">supercalifragilisticexpialidocious</span> will be split at the line break and
continue on the next line.
</div>
```

#div1: Small words are displayed normally, but a long word like **supercalifragilisticexpialidoc** is too long so it will overflow past the edge of the line-break

#div2: Small words are displayed normally, but a long word like **supercalifragilisticexpialidoci**ous will be split at the line break and continue on the next line.

overflow-wrap – Value	Details
normal	Lets a word overflow if it is longer than the line
break-word	Will split a word into multiple lines, if necessary
inherit	Inherits the parent element's value for this property

overflow: visible

HTML

```
<div>
  Even if this div is too small to display its contents, the content is not clipped.
</div>
```

CSS

```
div {
  width:50px;
  height:50px;
  overflow:visible;
}
```

Result



Content is not clipped and will be rendered outside the content box if it exceeds its container size.

Block Formatting Context Created with Overflow

Using the `overflow` property with a value different to `visible` will create a new **block formatting context**. This is useful for aligning a block element next to a floated element.

CSS

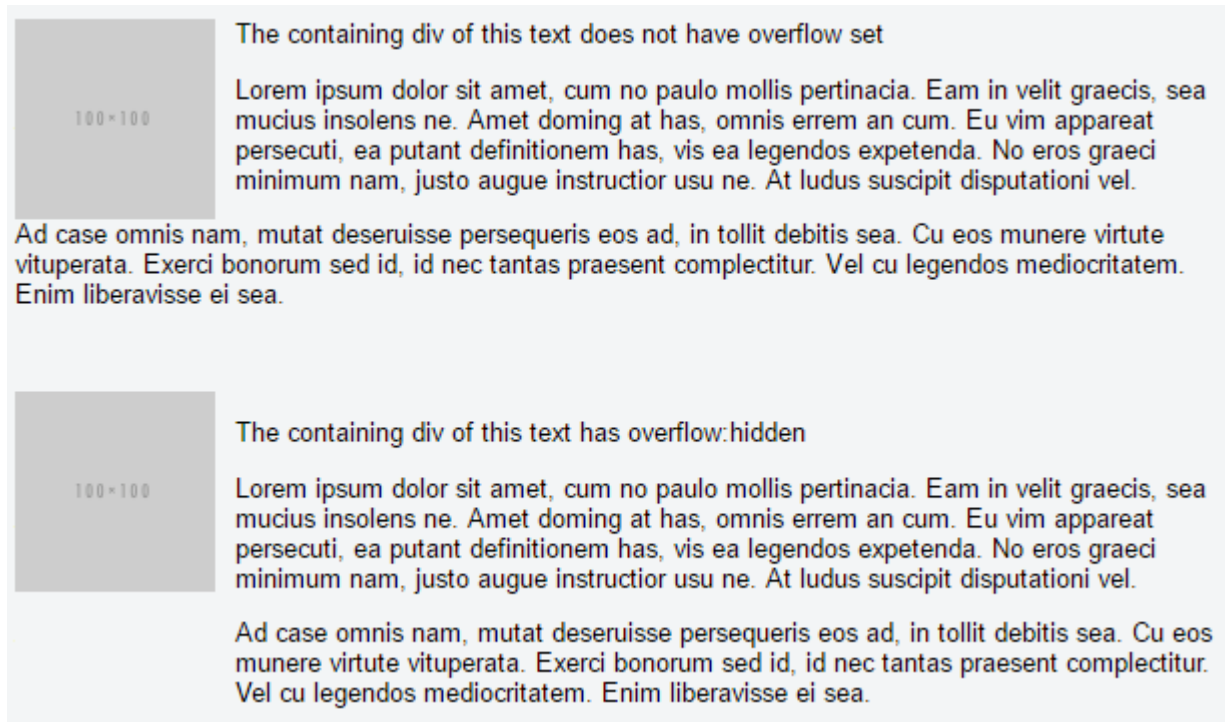
```
img {
  float:left;
  margin-right: 10px;
}
div {
  overflow:hidden; /* creates block formatting context */
}
```

HTML

```

<div>
  <p>Lorem ipsum dolor sit amet, cum no paulo mollis pertinacia.</p>
  <p>Ad case omnis nam, mutat deseruisse persequeris eos ad, in tollit debitis sea.</p>
</div>
```

Result



This example shows how paragraphs within a div with the `overflow` property set will interact with a floated image.

overflow-x and overflow-y

These two properties work in a similar fashion as the `overflow` property and accept the same values. The `overflow-x` parameter works only on the x or left-to-right axis. The `overflow-y` works on the y or top-to-bottom axis.

HTML

```
<div id="div-x">
  If this div is too small to display its contents,
  the content to the left and right will be clipped.
</div>

<div id="div-y">
  If this div is too small to display its contents,
  the content to the top and bottom will be clipped.
</div>
```

CSS

```
div {
  width: 200px;
  height: 200px;
}

#div-x {
  overflow-x: hidden;
}
```

```
#div-y {  
    overflow-y: hidden;  
}
```

Read Overflow online: <https://riptutorial.com/css/topic/4947/overflow>

Chapter 43: Padding

Syntax

- padding: *length*|initial|inherit|unset;
- padding-top: *length*|initial|inherit|unset;
- padding-right: *length*|initial|inherit|unset;
- padding-bottom: *length*|initial|inherit|unset;
- padding-left: *length*|initial|inherit|unset;

Remarks

The padding property sets the padding space on all sides of an element. The padding area is the space between the content of the element and its border. **Negative values are not allowed.**

1: <https://developer.mozilla.org/en/docs/Web/CSS/padding> MDN

Also see this [question](#), "Why does CSS not support negative padding?" and his answers.

Also please consider [The Box Model](#) when using padding. Depending on the box-sizing value, padding on an element can either add to the previously defined height/width of an element or not.

Related Properties:

[margin](#)

Padding on inline elements will only apply to the left and right of the element, and not the top and bottom, due to the inherent display properties of inline elements.

Examples

Padding on a given side

The padding property sets the padding space on all sides of an element. The padding area is the space between the content of the element and its border. Negative values are not allowed.

You can specify a side individually:

- padding-top
- padding-right
- padding-bottom
- padding-left

The following code would add a padding of 5px to the top of the div:

```
<style>
```



```
.myClass {
  padding-top: 5px;
}
</style>

<div class="myClass"></div>
```

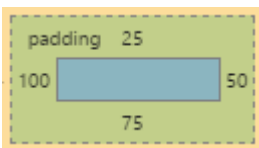
Padding Shorthand

The padding property sets the padding space on all sides of an element. The padding area is the space between the content of the element and its border. Negative values are not allowed.

To save adding padding to each side individually (using `padding-top`, `padding-left` etc) can you write it as a shorthand, as below:

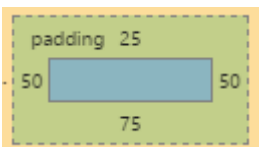
Four values:

```
<style>
  .myDiv {
    padding: 25px 50px 75px 100px; /* top right bottom left; */
  }
</style>
<div class="myDiv"></div>
```



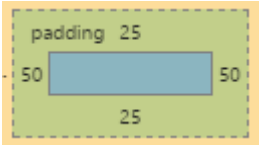
Three values:

```
<style>
  .myDiv {
    padding: 25px 50px 75px; /* top left/right bottom */
  }
</style>
<div class="myDiv"></div>
```



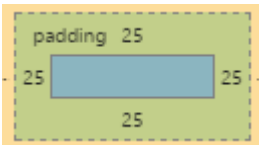
Two values:

```
<style>
  .myDiv {
    padding: 25px 50px; /* top/bottom left/right */
  }
</style>
<div class="myDiv"></div>
```



One value:

```
<style>
  .myDiv {
    padding: 25px; /* top/right/bottom/left */
  }
</style>
<div class="myDiv"></div>
```



Read Padding online: <https://riptutorial.com/css/topic/1255/padding>

Chapter 44: Performance

Examples

Use transform and opacity to avoid trigger layout

Changing some CSS attribute will trigger the browser to synchronously calculate the style and layout, which is a bad thing when you need to animate at 60fps.

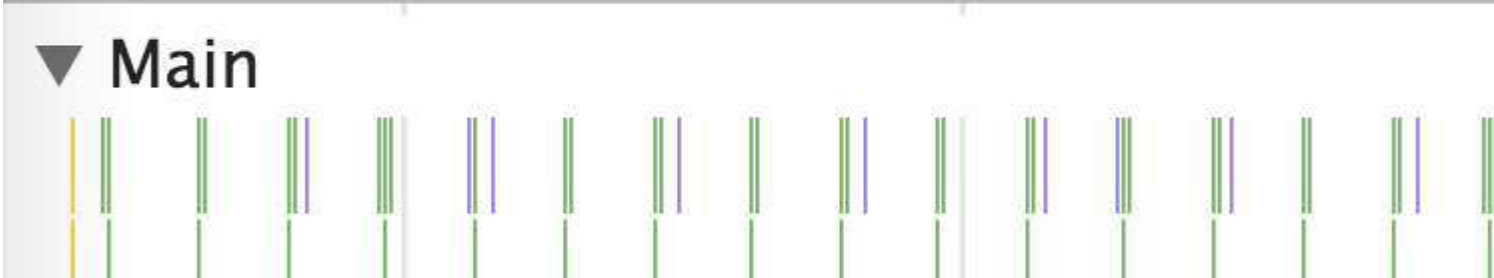
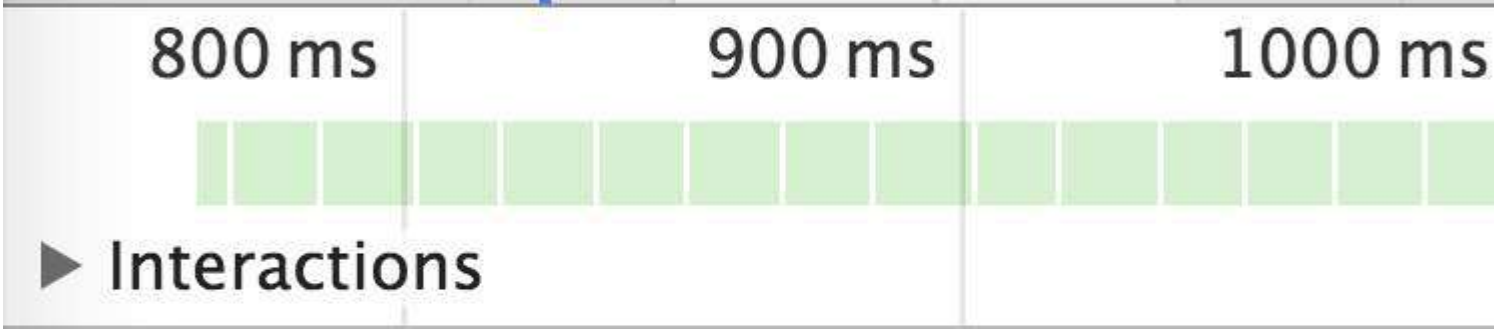
DON'T

Animate with `left` and `top` trigger layout.

```
#box {  
  left: 0;  
  top: 0;  
  transition: left 0.5s, top 0.5s;  
  position: absolute;  
  width: 50px;  
  height: 50px;  
  background-color: gray;  
}  
  
#box.active {  
  left: 100px;  
  top: 100px;  
}
```

Demo took **11.7ms** for rendering, **9.8ms** for painting

● | Capture: ☐ Network ☒ JS Pro



Summary Bottom-Up Call Tree Event Log

Group by Category ▼

Self Time ▼		Total Time		Active
11.7 ms	54.2 %	11.7 ms	54.2 %	▼
4.2 ms	19.5 %	4.2 ms	19.5 %	
3.9 ms	18.2 %	3.9 ms	18.2 %	
1.8 ms	8.3 %	1.8 ms	8.3 %	

Chapter 45: Positioning

Syntax

- position: static|absolute|fixed|relative|sticky|initial|inherit|unset;
- z-index: auto|*number*|initial|inherit;

Parameters

Parameter	Details
static	Default value. Elements render in order, as they appear in the document flow. The top, right, bottom, left and z-index properties do not apply.
relative	The element is positioned relative to its normal position, so <code>left:20px</code> adds 20 pixels to the element's LEFT position
fixed	The element is positioned relative to the browser window
absolute	The element is positioned relative to its first positioned (not static) ancestor element
initial	Sets this property to its default value.
inherit	Inherits this property from its parent element.
sticky	Experimental feature. It behaves like <code>position: static</code> within its parent until a given offset threshold is reached, then it acts as <code>position: fixed</code> .
unset	Combination of initial and inherit. More info here .

Remarks

Normal Flow is the flow of elements if the position of element is *static*.

1. defining *width* is beneficial because in some cases it prevents overlapping of element's content.

Examples

Fixed position

Defining position as fixed we can remove an element from the document flow and set its position relatively to the browser window. One obvious use is when we want something to be visible when

we scroll to the bottom of a long page.

```
#stickyDiv {  
  position:fixed;  
  top:10px;  
  left:10px;  
}
```

Overlapping Elements with z-index

To change the default [stack order](#) positioned elements (`position` property set to `relative`, `absolute` or `fixed`), use the `z-index` property.

The higher the z-index, the higher up in the stacking context (on the z-axis) it is placed.

Example

In the example below, a z-index value of 3 puts green on top, a z-index of 2 puts red just under it, and a z-index of 1 puts blue under that.

HTML

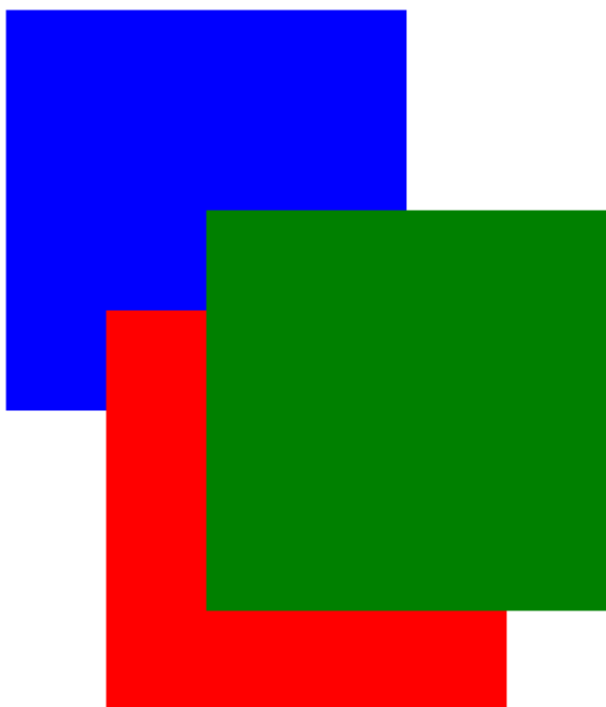
```
<div id="div1"></div>  
<div id="div2"></div>  
<div id="div3"></div>
```

CSS

```
div {  
  position: absolute;  
  height: 200px;  
  width: 200px;  
}  
div#div1 {  
  z-index: 1;  
  left: 0px;  
  top: 0px;  
  background-color: blue;  
}  
div#div2 {  
  z-index: 3;  
  left: 100px;  
  top: 100px;  
  background-color: green;  
}  
div#div3 {  
  z-index: 2;  
  left: 50px;  
  top: 150px;  
  background-color: red;
```

```
}
```

This creates the following effect:



See a working example at [JSFiddle](#).

Syntax

```
z-index: [ number ] | auto;
```

Parameter	Details
number	An integer value. A higher number is higher on the <code>z-index</code> stack. 0 is the default value. Negative values are allowed.
auto	Gives the element the same stacking context as its parent. (Default)

Remarks

All elements are laid out in a 3D axis in CSS, including a depth axis, measured by the `z-index` property. `z-index` only works on positioned elements: (see: [Why does z-index need a defined position to work?](#)). The only value where it is ignored is the default value, `static`.

Read about the `z-index` property and Stacking Contexts in the [CSS Specification](#) on layered presentation and at the [Mozilla Developer Network](#).

Relative Position

Relative positioning moves the element in relation to where it would have been in *normal flow*. Offset properties:

1. top
2. left
3. right
4. bottom

are used to indicate how far to move the element from where it would have been in normal flow.

```
.relpos{
  position:relative;
  top:20px;
  left:30px;
}
```

This code will move the box containing element with attribute `class="relpos"` 20px down and 30px to the right from where it would have been in normal flow.

Absolute Position

When absolute positioning is used the box of the desired element is taken out of the *Normal Flow* and it no longer affects the position of the other elements on the page. Offset properties:

1. top
2. left
3. right
4. bottom

specify the element should appear in relation to its next non-static containing element.

```
.abspos{
  position:absolute;
  top:0px;
  left:500px;
}
```

This code will move the box containing element with attribute `class="abspos"` down 0px and right 500px relative to its containing element.

Static positioning

The default position of an element is `static`. To quote [MDN](#):

This keyword lets the element use the normal behavior, that is it is laid out in its current position in the flow. The top, right, bottom, left and z-index properties do not apply.


```
.element{  
  position:static;  
}
```

Read Positioning online: <https://riptutorial.com/css/topic/935/positioning>

Chapter 46: Pseudo-Elements

Introduction

Pseudo-elements, just like pseudo-classes, are added to a CSS selectors but instead of describing a special state, they allow you to scope and style certain parts of an html element.

For example, the `::first-letter` pseudo-element targets only the first letter of a block element specified by the selector.

Syntax

- `selector::pseudo-element {property: value}`

Parameters

pseudo-element	Description
<code>::after</code>	Insert content after the content of an element
<code>::before</code>	Insert content before the content of an element
<code>::first-letter</code>	Selects the first letter of each element
<code>::first-line</code>	Selects the first line of each element
<code>::selection</code>	Matches the portion of an element that is selected by a user
<code>::backdrop</code>	Used to create a backdrop that hides the underlying document for an element in the top layer's stack
<code>::placeholder</code>	Allows you to style the placeholder text of a form element (Experimental)
<code>::marker</code>	For applying list-style attributes on a given element (Experimental)
<code>::spelling-error</code>	Represents a text segment which the browser has flagged as incorrectly spelled (Experimental)
<code>::grammar-error</code>	Represents a text segment which the browser has flagged as grammatically incorrect (Experimental)

Remarks

- Sometimes you will see double colons (`::`) instead of just one (`:`). This is a way to separate

pseudo-classes from pseudo-elements, but some older browsers like Internet Explorer 8 **only** supports single colon (:) for pseudo-elements.

- One can use only one pseudo-element in a selector. It must appear after the simple selectors in the statement.
- Pseudo-elements are not a part of the DOM, therefore are not targetable by `:hover` or other user events.

Examples

Pseudo-Elements

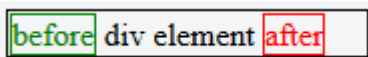
Pseudo-elements are added to selectors but instead of describing a special state, they allow you to style certain parts of a document.

The `content` attribute is required for pseudo-elements to render; however, the attribute can have an empty value (e.g. `content: ""`).

```
div::after {
  content: 'after';
  color: red;
  border: 1px solid red;
}

div {
  color: black;
  border: 1px solid black;
  padding: 1px;
}

div::before {
  content: 'before';
  color: green;
  border: 1px solid green;
}
```



Pseudo-Elements in Lists

Pseudo-elements are often used to change the look of lists (mostly for unordered lists, `ul`).

The first step is to remove the default list bullets:

```
ul {
  list-style-type: none;
}
```

Then you add the custom styling. In this example, we will create gradient boxes for bullets.

```
li:before {  
  content: "";  
  display: inline-block;  
  margin-right: 10px;  
  height: 10px;  
  width: 10px;  
  background: linear-gradient(red, blue);  
}
```

HTML

```
<ul>  
  <li>Test I</li>  
  <li>Test II</li>  
</ul>
```

Result



Read Pseudo-Elements online: <https://riptutorial.com/css/topic/911/pseudo-elements>

Chapter 47: Selectors

Introduction

CSS selectors identify specific HTML elements as targets for CSS styles. This topic covers how CSS selectors target HTML elements. Selectors use a wide range of over 50 selection methods offered by the CSS language, including elements, classes, IDs, pseudo-elements and pseudo-classes, and patterns.

Syntax

- *#id*
- *.classname*
- *:pseudo-classname*
- *::pseudo-elementname*
- *[attr]* /* has the *attr* attribute. */
- *[attr="value"]* /* has the *attr* attribute, and its value is exactly "value". */
- *[attr~="value"]* /* has the *attr* attribute, and its value, when split on **whitespace**, contains "value". */
- *[attr|= "value"]* /* has the *attr* attribute, and its value is exactly "value", **or** its value begins with "value". */
- *[attr^="value"]* /* has the *attr* attribute, and its value begins with "value". */
- *[attr\$="value"]* /* has the *attr* attribute, and its value ends with "value". */
- *[attr*="value"]* /* has the *attr* attribute, and its value contains "value". */
- *element-name*
- ***

Remarks

- Sometimes you will see double colons (: :) instead of just one (:). This is a way to separate [pseudo-classes](#) from [pseudo-elements](#).
- Old browsers, like Internet Explorer 8, **only** support a single colon (:) for defining pseudo-elements.
- Unlike pseudo-classes, only one pseudo-element may be used per selector, if present it must appear after the sequence of simple selectors that represents the subjects of the selector (a future version of the [W3C specification](#) may allow multiple pseudo-elements per selector).

Examples

Attribute Selectors

Overview

Attribute selectors can be used with various types of operators that change the selection criteria accordingly. They select an element using the presence of a given attribute or attribute value.

Selector ⁽¹⁾	Matched element	Selects elements...	CSS Version
[attr]	<div attr>	With attribute attr	2
[attr='val']	<div attr="val">	Where attribute attr has value val	2
[attr~='val']	<div attr="val val2 val3">	Where val appears in the whitespace-separated list of attr	2
[attr^='val']	<div attr="val1 val2">	Where attr's value <i>begins</i> with val	3
[attr\$='val']	<div attr="sth aval">	Where the attr's value <i>ends</i> with val	3
[attr*='val']	<div attr="somevalhere">	Where attr contains val anywhere	3
[attr ='val']	<div attr="val-sth etc">	Where attr's value is exactly val, or starts with val and immediately followed by - (U+002D)	2
[attr='val' i]	<div attr="val">	Where attr has value val, ignoring val's letter casing.	4 ⁽²⁾

Notes:

1. The attribute value can be surrounded by either single-quotes or double-quotes. No quotes at all may also work, but it's not valid according to the CSS standard, and is discouraged.
2. There is no single, integrated CSS4 specification, because it is split into separate modules. However, there are "level 4" modules. [See browser support](#).

Details

[attribute]

Selects elements with the given attribute.

```
div[data-color] {
```

```
color: red;
}
```

```
<div data-color="red">This will be red</div>
<div data-color="green">This will be red</div>
<div data-background="red">This will NOT be red</div>
```

[Live Demo on JSBin](#)

[attribute="value"]

Selects elements with the given attribute and value.

```
div[data-color="red"] {
  color: red;
}
```

```
<div data-color="red">This will be red</div>
<div data-color="green">This will NOT be red</div>
<div data-color="blue">This will NOT be red</div>
```

[Live Demo on JSBin](#)

[attribute*="value"]

Selects elements with the given attribute and value where the given attribute contains the given value anywhere (as a substring).

```
[class*="foo"] {
  color: red;
}
```

```
<div class="foo-123">This will be red</div>
<div class="foo123">This will be red</div>
<div class="bar123foo">This will be red</div>
<div class="barfoo123">This will be red</div>
<div class="barfo0">This will NOT be red</div>
```

[Live Demo on JSBin](#)

[attribute~="value"]

Selects elements with the given attribute and value where the given value appears in a whitespace-separated list.

```
[class~="color-red"] {
  color: red;
}
```

```
<div class="color-red foo-bar the-div">This will be red</div>
<div class="color-blue foo-bar the-div">This will NOT be red</div>
```

[Live Demo on JSBin](#)

[attribute^="value"]

Selects elements with the given attribute and value where the given attribute begins with the value.

```
[class^="foo-"] {  
  color: red;  
}
```

```
<div class="foo-123">This will be red</div>  
<div class="foo-234">This will be red</div>  
<div class="bar-123">This will NOT be red</div>
```

[Live Demo on JSBin](#)

[attribute\$="value"]

Selects elements with the given attribute and value where the given attribute ends with the given value.

```
[class$="file"] {  
  color: red;  
}
```

```
<div class="foobar-file">This will be red</div>  
<div class="foobar-file">This will be red</div>  
<div class="foobar-input">This will NOT be red</div>
```

[Live Demo on JSBin](#)

[attribute|="value"]

Selects elements with a given attribute and value where the attribute's value is exactly the given value or is exactly the given value followed by - (U+002D)

```
[lang|="EN"] {  
  color: red;  
}
```

```
<div lang="EN-us">This will be red</div>  
<div lang="EN-gb">This will be red</div>  
<div lang="PT-pt">This will NOT be red</div>
```

[Live Demo on JSBin](#)

[attribute="value" i]

Selects elements with a given attribute and value where the attribute's value can be represented

as `Value`, `VALUE`, `vaLue` or any other case-insensitive possibility.

```
[lang="EN" i] {  
  color: red;  
}
```

```
<div lang="EN">This will be red</div>  
<div lang="en">This will be red</div>  
<div lang="PT">This will NOT be red</div>
```

[Live Demo on JSBin](#)

Specificity of attribute selectors

0-1-0

Same as class selector and pseudoclass.

```
*[type=checkbox] // 0-1-0
```

Note that this means an attribute selector can be used to select an element by its ID at a lower level of specificity than if it was selected with an [ID selector](#): `[id="my-ID"]` targets the same element as `#my-ID` but with lower specificity.

See the [Syntax Section](#) for more details.

Combinators

Overview

Selector	Description
<code>div span</code>	Descendant selector (all <code></code> s that are descendants of a <code><div></code>)
<code>div > span</code>	Child selector (all <code></code> s that are a direct child of a <code><div></code>)
<code>a ~ span</code>	General Sibling selector (all <code></code> s that are siblings after an <code><a></code>)
<code>a + span</code>	Adjacent Sibling selector (all <code></code> s that are immediately after an <code><a></code>)

Note: Sibling selectors target elements that come after them in the source document. CSS, by its nature (it cascades), cannot target *previous* or *parent* elements. However, using the `flex order` property, [a previous sibling selector can be simulated on visual media](#).

Descendant Combinator: `selector selector`

A descendant combinator, represented by at least one space character (), selects elements that are a descendant of the defined element. This combinator selects **all** descendants of the element (from child elements on down).

```
div p {  
  color:red;  
}
```

```
<div>  
  <p>My text is red</p>  
  <section>  
    <p>My text is red</p>  
  </section>  
</div>  
  
<p>My text is not red</p>
```

[Live Demo on JSBin](#)

In the above example, the first two `<p>` elements are selected since they are both descendants of the `<div>`.

Child Combinator: `selector > selector`

The child (`>`) combinator is used to select elements that are **children**, or **direct descendants**, of the specified element.

```
div > p {  
  color:red;  
}
```

```
<div>  
  <p>My text is red</p>  
  <section>  
    <p>My text is not red</p>  
  </section>  
</div>
```

[Live Demo on JSBin](#)

The above CSS selects only the first `<p>` element, as it is the only paragraph directly descended from a `<div>`.

The second `<p>` element is not selected because it is not a direct child of the `<div>`.

Adjacent Sibling Combinator: `selector + selector`

The adjacent sibling (+) combinator selects a sibling element that immediately follows a specified element.

```
p + p {  
  color:red;  
}
```

```
<p>My text is not red</p>  
<p>My text is red</p>  
<p>My text is red</p>  
<hr>  
<p>My text is not red</p>
```

[Live Demo on JSBin](#)

The above example selects only those `<p>` elements which are *directly preceded* by another `<p>` element.

General Sibling Combinator: `selector ~ selector`

The general sibling (~) combinator selects *all* siblings that follow the specified element.

```
p ~ p {  
  color:red;  
}
```

```
<p>My text is not red</p>  
<p>My text is red</p>  
<hr>  
<h1>And now a title</h1>  
<p>My text is red</p>
```

[Live Demo on JSBin](#)

The above example selects all `<p>` elements that are *preceded* by another `<p>` element, whether or not they are immediately adjacent.

Class Name Selectors

The class name selector selects all elements with the targeted class name. For example, the class name `.warning` would select the following `<div>` element:

```
<div class="warning">  
  <p>This would be some warning copy.</p>  
</div>
```

You can also combine class names to target elements more specifically. Let's build on the example above to showcase a more complicated class selection.

CSS

```
.important {
  color: orange;
}
.warning {
  color: blue;
}
.warning.important {
  color: red;
}
```

HTML

```
<div class="warning">
  <p>This would be some warning copy.</p>
</div>

<div class="important warning">
  <p class="important">This is some really important warning copy.</p>
</div>
```

In this example, all elements with the `.warning` class will have a blue text color, elements with the `.important` class will have an orange text color, and all elements that have *both* the `.important` and `.warning` class name will have a red text color.

Notice that within the CSS, the `.warning.important` declaration did not have any spaces between the two class names. This means it will only find elements which contain both class names `warning` and `important` in their `class` attribute. Those class names could be in any order on the element.

If a space was included between the two classes in the CSS declaration, it would only select elements that have parent elements with a `.warning` class name and child elements with `.important` class names.

ID selectors

ID selectors select DOM elements with the targeted ID. To select an element by a specific ID in CSS, the `#` prefix is used.

For example, the following HTML `div` element...

```
<div id="exampleID">
  <p>Example</p>
</div>
```

...can be selected by `#exampleID` in CSS as shown below:

```
#exampleID {
```

```
width: 20px;
}
```

Note: The HTML specs do not allow multiple elements with the same ID

Pseudo-classes

Pseudo-classes are **keywords** which allow selection based on information that lies outside of the document tree or that cannot be expressed by other selectors or combinators. This information can be associated to a certain state (**state** and **dynamic** pseudo-classes), to locations (**structural** and **target** pseudo-classes), to negations of the former (**negation** pseudo-class) or to languages (**lang** pseudo-class). Examples include whether or not a link has been followed (`:visited`), the mouse is over an element (`:hover`), a checkbox is checked (`:checked`), etc.

Syntax

```
selector:pseudo-class {
  property: value;
}
```

List of pseudo-classes:

Name	Description
<code>:active</code>	Applies to any element being activated (i.e. clicked) by the user.
<code>:any</code>	Allows you to build sets of related selectors by creating groups that the included items will match. This is an alternative to repeating an entire selector.
<code>:target</code>	Selects the current active <code>#news</code> element (clicked on a URL containing that anchor name)
<code>:checked</code>	Applies to radio, checkbox, or option elements that are checked or toggled into an "on" state.
<code>:default</code>	Represents any user interface element that is the default among a group of similar elements.
<code>:disabled</code>	Applies to any UI element which is in a disabled state.
<code>:empty</code>	Applies to any element which has no children.
<code>:enabled</code>	Applies to any UI element which is in an enabled state.
<code>:first</code>	Used in conjunction with the <code>@page</code> rule, this selects the first page in a printed document.

Name	Description
<code>:first-child</code>	Represents any element that is the first child element of its parent.
<code>:first-of-type</code>	Applies when an element is the first of the selected element type inside its parent. This may or may not be the first-child.
<code>:focus</code>	Applies to any element which has the user's focus. This can be given by the user's keyboard, mouse events, or other forms of input.
<code>:focus-within</code>	Can be used to highlight a whole section when one element inside it is focused. It matches any element that the <code>:focus</code> pseudo-class matches or that has a descendant focused.
<code>:full-screen</code>	Applies to any element displayed in full-screen mode. It selects the whole stack of elements and not just the top level element.
<code>:hover</code>	Applies to any element being hovered by the user's pointing device, but not activated.
<code>:indeterminate</code>	Applies radio or checkbox UI elements which are neither checked nor unchecked, but are in an indeterminate state. This can be due to an element's attribute or DOM manipulation.
<code>:in-range</code>	The <code>:in-range</code> CSS pseudo-class matches when an element has its value attribute inside the specified range limitations for this element. It allows the page to give a feedback that the value currently defined using the element is inside the range limits.
<code>:invalid</code>	Applies to <code><input></code> elements whose values are invalid according to the type specified in the <code>type=</code> attribute.
<code>:lang</code>	Applies to any element who's wrapping <code><body></code> element has a properly designated <code>lang=</code> attribute. For the pseudo-class to be valid, it must contain a valid two or three letter language code .
<code>:last-child</code>	Represents any element that is the last child element of its parent.
<code>:last-of-type</code>	Applies when an element is the last of the selected element type inside its parent. This may or may not be the last-child.
<code>:left</code>	Used in conjunction with the <code>@page</code> rule, this selects all the left pages in a printed document.
<code>:link</code>	Applies to any links which haven't been visited by the user.
<code>:not()</code>	Applies to all elements which do not match the value passed to <code>(:not(p) or :not(.class-name))</code> for example. It must have a value to be

Name	Description
	valid and it can only contain one selector. However, you can chain multiple <code>:not</code> selectors together.
<code>:nth-child</code>	Applies when an element is the n -th element of its parent, where n can be an integer, a mathematical expression (e.g $n+3$) or the keywords <code>odd</code> or <code>even</code> .
<code>:nth-of-type</code>	Applies when an element is the n -th element of its parent of the same element type, where n can be an integer, a mathematical expression (e.g $n+3$) or the keywords <code>odd</code> or <code>even</code> .
<code>:only-child</code>	The <code>:only-child</code> CSS pseudo-class represents any element which is the only child of its parent. This is the same as <code>:first-child:last-child</code> or <code>:nth-child(1):nth-last-child(1)</code> , but with a lower specificity.
<code>:optional</code>	The <code>:optional</code> CSS pseudo-class represents any element that does not have the required attribute set on it. This allows forms to easily indicate optional fields and to style them accordingly.
<code>:out-of-range</code>	The <code>:out-of-range</code> CSS pseudo-class matches when an element has its value attribute outside the specified range limitations for this element. It allows the page to give a feedback that the value currently defined using the element is outside the range limits. A value can be outside of a range if it is either smaller or larger than maximum and minimum set values.
<code>:placeholder-shown</code>	Experimental. Applies to any form element currently displaying placeholder text.
<code>:read-only</code>	Applies to any element which is not editable by the user.
<code>:read-write</code>	Applies to any element that is editable by a user, such as <code><input></code> elements.
<code>:right</code>	Used in conjunction with the <code>@page</code> rule, this selects all the right pages in a printed document.
<code>:root</code>	matches the root element of a tree representing the document.
<code>:scope</code>	CSS pseudo-class matches the elements that are a reference point for selectors to match against.
<code>:target</code>	Selects the current active <code>#news</code> element (clicked on a URL containing that anchor name)
<code>:visited</code>	Applies to any links which have has been visited by the user.

The `:visited` pseudoclass can't be used for most styling in a lot of modern browsers anymore because it's a security hole. See this [link](#) for reference.

Basic selectors

Selector	Description
*	Universal selector (all elements)
div	Tag selector (all <code><div></code> elements)
.blue	Class selector (all elements with class <code>blue</code>)
.blue.red	All elements with class <code>blue</code> and <code>red</code> (a type of Compound selector)
#headline	ID selector (the element with "id" attribute set to <code>headline</code>)
:pseudo-class	All elements with pseudo-class
::pseudo-element	Element that matches pseudo-element
:lang(en)	Element that matches <code>:lang</code> declaration, for example <code></code>
div > p	child selector

Note: The value of an ID must be unique in a web page. It is a violation of the [HTML standard](#) to use the value of an ID more than once in the same document tree.

A complete list of selectors can be found in the [CSS Selectors Level 3 specification](#).

How to style a Range input

HTML

```
<input type="range"></input>
```

CSS

Effect	Pseudo Selector
Thumb	<code>input[type=range]:-webkit-slider-thumb, input[type=range]:-moz-range-thumb, input[type=range]:-ms-thumb</code>
Track	<code>input[type=range]:-webkit-slider-runnable-track, input[type=range]:-moz-range-track, input[type=range]:-ms-track</code>
OnFocus	<code>input[type=range]:focus</code>
Lower part of the track	<code>input[type=range]:-moz-range-progress, input[type=range]:-ms-fill-lower</code> (not possible in WebKit browsers currently - JS needed)

Global boolean with checkbox:checked and ~ (general sibling combinator)

With the ~ selector, you can easily implement a global accessible boolean without using JavaScript.

Add boolean as a checkbox

To the very beginning of your document, add as much booleans as you want with a unique `id` and the `hidden` attribute set:

```
<input type="checkbox" id="sidebarShown" hidden />
<input type="checkbox" id="darkThemeUsed" hidden />

<!-- here begins actual content, for example: -->
<div id="container">
  <div id="sidebar">
    <!-- Menu, Search, ... -->
  </div>

  <!-- Some more content ... -->
</div>

<div id="footer">
  <!-- ... -->
</div>
```

Change the boolean's value

You can toggle the boolean by adding a `label` with the `for` attribute set:

```
<label for="sidebarShown">Show/Hide the sidebar!</label>
```

Accessing boolean value with CSS

The normal selector (like `.color-red`) specifies the default properties. They can be overridden by following `true` / `false` selectors:

```
/* true: */
<checkbox>:checked ~ [sibling of checkbox & parent of target] <target>

/* false: */
<checkbox>:not(:checked) ~ [sibling of checkbox & parent of target] <target>
```

Note that `<checkbox>`, `[sibling ...]` and `<target>` should be replaced by the proper selectors. `[sibling ...]` can be a specific selector, (often if you're lazy) simply `*` or nothing if the target is already a sibling of the checkbox.

Examples for the above HTML structure would be:

```
#sidebarShown:checked ~ #container #sidebar {
  margin-left: 300px;
}

#darkThemeUsed:checked ~ #container,
#darkThemeUsed:checked ~ #footer {
  background: #333;
}
```

In action

See [this fiddle](#) for a implementation of these global booleans.

CSS3 :in-range selector example

```
<style>
input:in-range {
  border: 1px solid blue;
}
</style>

<input type="number" min="10" max="20" value="15">
<p>The border for this value will be blue</p>
```

The `:in-range` CSS pseudo-class matches when an element has its value attribute inside the specified range limitations for this element. It allows the page to give a feedback that the value currently defined using the element is inside the range limits.[\[1\]](#)

Child Pseudo Class

"The `:nth-child(an+b)` CSS pseudo-class matches an element that has $an+b-1$ siblings before it in the document tree, for a given positive **or zero value** for n " - [MDN :nth-child](#)

pseudo-selector	1	2	3	4	5	6	7	8	9	10
:first-child	✓									
:nth-child(3)			✓							
:nth-child(n+3)			✓	✓	✓	✓	✓	✓	✓	✓
:nth-child(3n)			✓			✓			✓	
:nth-child(3n+1)	✓			✓			✓			✓
:nth-child(-n+3)	✓	✓	✓							

pseudo-selector	1	2	3	4	5	6	7	8	9	10
:nth-child(odd)	✓		✓		✓		✓		✓	
:nth-child(even)		✓		✓		✓		✓		✓
:last-child										✓
:nth-last-child(3)								✓		

Select element using its ID without the high specificity of the ID selector

This trick helps you select an element using the ID as a value for an attribute selector to avoid the high specificity of the ID selector.

HTML:

```
<div id="element">...</div>
```

CSS

```
#element { ... } /* High specificity will override many selectors */
[id="element"] { ... } /* Low specificity, can be overridden easily */
```

A. The :not pseudo-class example & B. :focus-within CSS pseudo-class

A. The syntax is presented above.

The following selector matches all `<input>` elements in an HTML document that are not disabled and don't have the class `.example`:

HTML:

```
<form>
  Phone: <input type="tel" class="example">
  E-mail: <input type="email" disabled="disabled">
  Password: <input type="password">
</form>
```

CSS:

```
input:not([disabled]):not(.example){
  background-color: #ccc;
}
```

The `:not()` pseudo-class will also support comma-separated selectors in Selectors Level 4:

CSS:

```
input:not([disabled], .example){
  background-color: #ccc;
}
```

[Live Demo on JSBin](#)

See background syntax [here](#).

B. The :focus-within CSS pseudo-class

HTML:

```
<h3>Background is blue if the input is focused .</p>
<div>
  <input type="text">
</div>
```

CSS:

```
div {
  height: 80px;
}
input{
  margin:30px;
}
div:focus-within {
  background-color: #1565C0;
}
```

```
div {
  height: 80px;
}
input{
  margin:30px;
}
div:focus-within {
  background-color: #1565C0;
}
```

Background is blue if the input is focused .



#

:focus-within CSS pseudo-class 📄 - UNOFF

The **:focus-within** pseudo-class matches elements that either themselves match **:focus** or that have descendants which match **:focus**.

Current aligned

Usage relative

Date relative

Show all

IE	Edge	Firefox	Chrome	Safari	Op
		52	49		
	14	53	58		4
11	15	54	59	10.1	4
	16	55	60	11	4
		56	61	TP	4
		57	62		4

Notes

Known issues (0)

Resources (11)

Feedback

¹ Can be enabled via the "Experimental Web Platform Features" flag

The :only-child pseudo-class selector example

The **:only-child** CSS pseudo-class represents any element which is the only child of its parent.

HTML:

```
<div>
  <p>This paragraph is the only child of the div, it will have the color blue</p>
</div>

<div>
  <p>This paragraph is one of the two children of the div</p>
  <p>This paragraph is one of the two children of its parent</p>
</div>
```

CSS:

```
p:only-child {
```

```
color: blue;
}
```

The above example selects the `<p>` element that is the unique child from its parent, in this case a `<div>`.

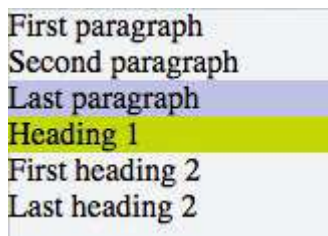
[Live Demo on JSBin](#)

The `:last-of-type` selector

The `:last-of-type` selects the element that is the last child, of a particular type, of its parent. In the example below, the css selects the last paragraph and the last heading `h1`.

```
p:last-of-type {
  background: #C5CAE9;
}
h1:last-of-type {
  background: #CDDC39;
}
```

```
<div class="container">
  <p>First paragraph</p>
  <p>Second paragraph</p>
  <p>Last paragraph</p>
  <h1>Heading 1</h1>
  <h2>First heading 2</h2>
  <h2>Last heading 2</h2>
</div>
```



First paragraph
Second paragraph
Last paragraph
Heading 1
First heading 2
Last heading 2

[jsFiddle](#)

Read Selectors online: <https://riptutorial.com/css/topic/611/selectors>

Chapter 48: Shapes for Floats

Syntax

- `shape-outside: none | [ <basic-shape> || <shape-box> ] | <image>`
- `shape-margin: <length> | <percentage>`
- `shape-image-threshold: <number>`

Parameters

Parameter	Details
<code>none</code>	A value of <code>none</code> means that the float area (the area that is used for wrapping content around a float element) is unaffected. This is the default/initial value.
<code>basic-shape</code>	Refers to one among <code>inset()</code> , <code>circle()</code> , <code>ellipse()</code> or <code>polygon()</code> . Using one of these functions and its values the shape is defined.
<code>shape-box</code>	Refers to one among <code>margin-box</code> , <code>border-box</code> , <code>padding-box</code> , <code>content-box</code> . When only <code><shape-box></code> is provided (without <code><basic-shape></code>) this box <i>is the</i> shape. When it is used along with <code><basic-shape></code> , this acts as the reference box.
<code>image</code>	When an image is provided as value, the shape is computed based on the alpha channel of the image specified.

Remarks

Browser support for the CSS Shapes module is very limited at this point in time.

It is supported in Chrome v37+ and Opera 24+ without browser/vendor prefixes. Safari supports it from v7.1+ but with the `-webkit-` prefix.

It is not yet supported in IE, Edge and Firefox.

Examples

Shape Outside with Basic Shape – `circle()`

With the `shape-outside` CSS property one can define shape values for the float area so that the inline content wraps around the shape instead of the float's box.

CSS

```
img:nth-of-type(1) {
  shape-outside: circle(80px at 50% 50%);
  float: left;
  width: 200px;
}
img:nth-of-type(2) {
  shape-outside: circle(80px at 50% 50%);
  float: right;
  width: 200px;
}
p {
  text-align: center;
  line-height: 30px; /* purely for demo */
}
```

HTML

```

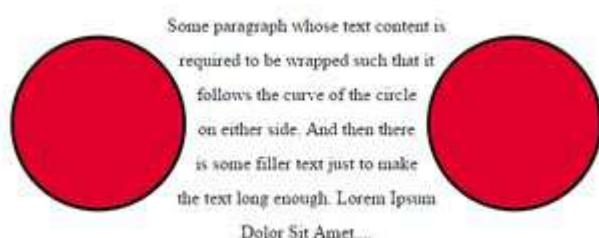

<p>Some paragraph whose text content is required to be wrapped such that it follows the curve
of the circle on either side. And then there is some filler text just to make the text long
enough. Lorem Ipsum Dolor Sit Amet....</p>
```

In the above example, both the images are actually square images and when the text is placed without the `shape-outside` property, it will not flow around the circle on either side. It will flow around the containing box of the image only. With `shape-outside` the float area is re-defined as a *circle* and the content is made to flow around this *imaginary circle* that is created using `shape-outside`.

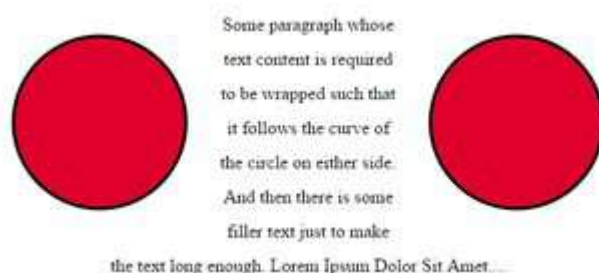
The *imaginary circle* that is used to re-define the float area is a circle with radius of 80px drawn from the center-mid point of the image's reference box.

Below are a couple of screenshots to illustrate how the content would be wrapped around when `shape-outside` is used and when it is not used.

Output with `shape-outside`



Output without `shape-outside`



Shape margin

The `shape-margin` CSS property adds a *margin* to `shape-outside`.

CSS

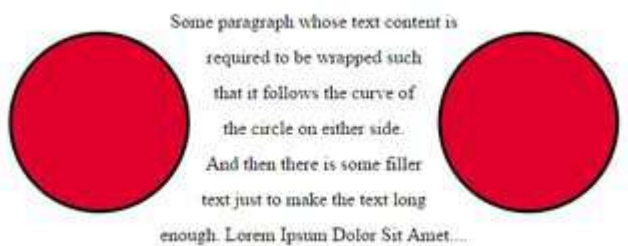
```
img:nth-of-type(1) {  
  shape-outside: circle(80px at 50% 50%);  
  shape-margin: 10px;  
  float: left;  
  width: 200px;  
}  
img:nth-of-type(2) {  
  shape-outside: circle(80px at 50% 50%);  
  shape-margin: 10px;  
  float: right;  
  width: 200px;  
}  
p {  
  text-align: center;  
  line-height: 30px; /* purely for demo */  
}
```

HTML

```
  
  
<p>Some paragraph whose text content is required to be wrapped such that it follows the curve  
of the circle on either side. And then there is some filler text just to make the text long  
enough. Lorem Ipsum Dolor Sit Amet....</p>
```

In this example, a 10px margin is added around the **shape** using `shape-margin`. This creates a bit more space between the *imaginary circle* that defines the float area and the actual content that is flowing around.

Output:



Read Shapes for Floats online: <https://riptutorial.com/css/topic/2034/shapes-for-floats>

Chapter 49: Single Element Shapes

Examples

Square

To create a square, define an element with both a width and height. In the example below, we have an element with a `width` and `height` of 100 pixels each.



```
<div class="square"></div>
```

```
.square {  
  width: 100px;  
  height: 100px;  
  background: rgb(246, 156, 85);  
}
```

Triangles

To create a CSS triangle define an element with a width and height of 0 pixels. The triangle shape will be formed using border properties. For an element with 0 height and width the 4 borders (top, right, bottom, left) each form a triangle. Here's an element with 0 height/width and 4 different colored borders.



By setting some borders to transparent, and others to a color we can create various triangles. For example, in the Up triangle, we set the bottom border to the desired color, then set the left and right borders to transparent. Here's an image with the left and right borders shaded slightly to show how the triangle is being formed.



The dimensions of the triangle can be altered by changing the different border widths - taller, shorter, lopsided, etc. The examples below all show a 50x50 pixel triangle.

Triangle - Pointing Up



```
<div class="triangle-up"></div>
```

```
.triangle-up {  
  width: 0;  
  height: 0;  
  border-left: 25px solid transparent;  
  border-right: 25px solid transparent;  
  border-bottom: 50px solid rgb(246, 156, 85);  
}
```

Triangle - Pointing Down



```
<div class="triangle-down"></div>
```

```
.triangle-down {  
  width: 0;  
  height: 0;  
  border-left: 25px solid transparent;  
  border-right: 25px solid transparent;  
  border-top: 50px solid rgb(246, 156, 85);  
}
```

Triangle - Pointing Right



```
<div class="triangle-right"></div>
```

```
.triangle-right {  
  width: 0;  
  height: 0;  
  border-top: 25px solid transparent;  
  border-bottom: 25px solid transparent;  
  border-left: 50px solid rgb(246, 156, 85);  
}
```

Triangle - Pointing Left



```
<div class="triangle-left"></div>
```

```
.triangle-left {  
  width: 0;  
  height: 0;  
  border-top: 25px solid transparent;  
  border-bottom: 25px solid transparent;  
  border-right: 50px solid rgb(246, 156, 85);  
}
```

Triangle - Pointing Up/Right



```
<div class="triangle-up-right"></div>
```

```
.triangle-up-right {  
  width: 0;  
  height: 0;  
  border-top: 50px solid rgb(246, 156, 85);  
  border-left: 50px solid transparent;  
}
```

Triangle - Pointing Up/Left



```
<div class="triangle-up-left"></div>
```

```
.triangle-up-left {  
  width: 0;  
  height: 0;  
  border-top: 50px solid rgb(246, 156, 85);  
  border-right: 50px solid transparent;  
}
```

Triangle - Pointing Down/Right



```
<div class="triangle-down-right"></div>
```

```
.triangle-down-right {  
  width: 0;  
  height: 0;  
  border-bottom: 50px solid rgb(246, 156, 85);  
  border-left: 50px solid transparent;  
}
```

Triangle - Pointing Down/Left



```
<div class="triangle-down-left"></div>
```

```
.triangle-down-left {  
  width: 0;  
  height: 0;  
  border-bottom: 50px solid rgb(246, 156, 85);  
  border-right: 50px solid transparent;  
}
```

```
}
```

Bursts

A burst is similar to a star but with the points extending less distance from the body. Think of a burst shape as a square with additional, slightly rotated, squares layered on top.

The additional squares are created using the `::before` and `::after` pseudo-elements.

8 Point Burst

An 8 point burst are 2 layered squares. The bottom square is the element itself, the additional square is created using the `:before` pseudo-element. The bottom is rotated 20°, the top square is rotated 135°.



```
<div class="burst-8"></div>

.burst-8 {
  background: rgb(246, 156, 85);
  width: 40px;
  height: 40px;
  position: relative;
  text-align: center;
  -ms-transform: rotate(20deg);
  transform: rotate(20deg);
}

.burst-8::before {
  content: "";
  position: absolute;
  top: 0;
  left: 0;
  height: 40px;
  width: 40px;
  background: rgb(246, 156, 85);
  -ms-transform: rotate(135deg);
  transform: rotate(135deg);
}
```

12 Point Burst

An 12 point burst are 3 layered squares. The bottom square is the element itself, the additional squares are created using the `:before` and `:after` pseudo-elements. The bottom is rotated 0°, the next square is rotated 30°, and the top is rotated 60°.



```
<div class="burst-12"></div>

.burst-12 {
  width: 40px;
  height: 40px;
  position: relative;
  text-align: center;
  background: rgb(246, 156, 85);
}

.burst-12::before, .burst-12::after {
  content: "";
  position: absolute;
  top: 0;
  left: 0;
  height: 40px;
  width: 40px;
  background: rgb(246, 156, 85);
}

.burst-12::before {
  -ms-transform: rotate(30deg);
  transform: rotate(30deg);
}

.burst-12::after {
  -ms-transform: rotate(60deg);
  transform: rotate(60deg);
}
```

Circles and Ellipses

Circle

To create a **circle**, define an element with an equal `width` and `height` (a *square*) and then set the `border-radius` property of this element to 50%.



HTML

```
<div class="circle"></div>
```

CSS

```
.circle {  
  width: 50px;  
  height: 50px;  
  background: rgb(246, 156, 85);  
  border-radius: 50%;  
}
```

Ellipse

An **ellipse** is similar to a circle, but with different values for `width` and `height`.



HTML

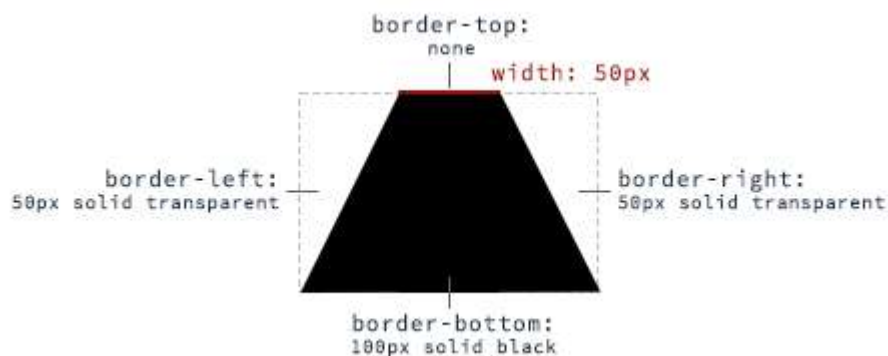
```
<div class="oval"></div>
```

CSS

```
.oval {  
  width: 50px;  
  height: 80px;  
  background: rgb(246, 156, 85);  
  border-radius: 50%;  
}
```

Trapezoid

A trapezoid can be made by a block element with zero height (height of `0px`), a width greater than zero and a border, that is transparent except for one side:



HTML:


```
<div class="trapezoid"></div>
```

CSS:

```
.trapezoid {  
  width: 50px;  
  height: 0;  
  border-left: 50px solid transparent;  
  border-right: 50px solid transparent;  
  border-bottom: 100px solid black;  
}
```

With changing the border sides, the orientation of the trapezoid can be adjusted.

Cube

This example shows how to create a cube using 2D transformation methods `skewX()` and `skewY()` on pseudo elements.



HTML:

```
<div class="cube"></div>
```

CSS:

```
.cube {  
  background: #dc2e2e;  
  width: 100px;  
  height: 100px;  
  position: relative;  
  margin: 50px;  
}  
  
.cube::before {  
  content: '';  
  display: inline-block;  
  background: #f15757;  
  width: 100px;  
  height: 20px;  
  transform: skewX(-40deg);  
  position: absolute;  
  top: -20px;  
  left: 8px;  
}
```

```

}

.cube::after {
  content: '';
  display: inline-block;
  background: #9e1515;
  width: 16px;
  height: 100px;
  transform: skewY(-50deg);
  position: absolute;
  top: -10px;
  left: 100%;
}

```

[See demo](#)

Pyramid

This example shows how to create a **pyramid** using borders and 2D transformation methods `skewY()` and `rotate()` on pseudo elements.



HTML:

```
<div class="pyramid"></div>
```

CSS:

```

.pyramid {
  width: 100px;
  height: 200px;
  position: relative;
  margin: 50px;
}

.pyramid::before, .pyramid::after {
  content: '';
  display: inline-block;
  width: 0;
  height: 0;
  border: 50px solid;
  position: absolute;
}

.pyramid::before {
  border-color: transparent transparent #ff5656 transparent;
}

```

```
transform: scaleY(2) skewY(-40deg) rotate(45deg);  
}  
  
.pyramid::after {  
border-color: transparent transparent #d64444 transparent;  
transform: scaleY(2) skewY(40deg) rotate(-45deg);  
}
```

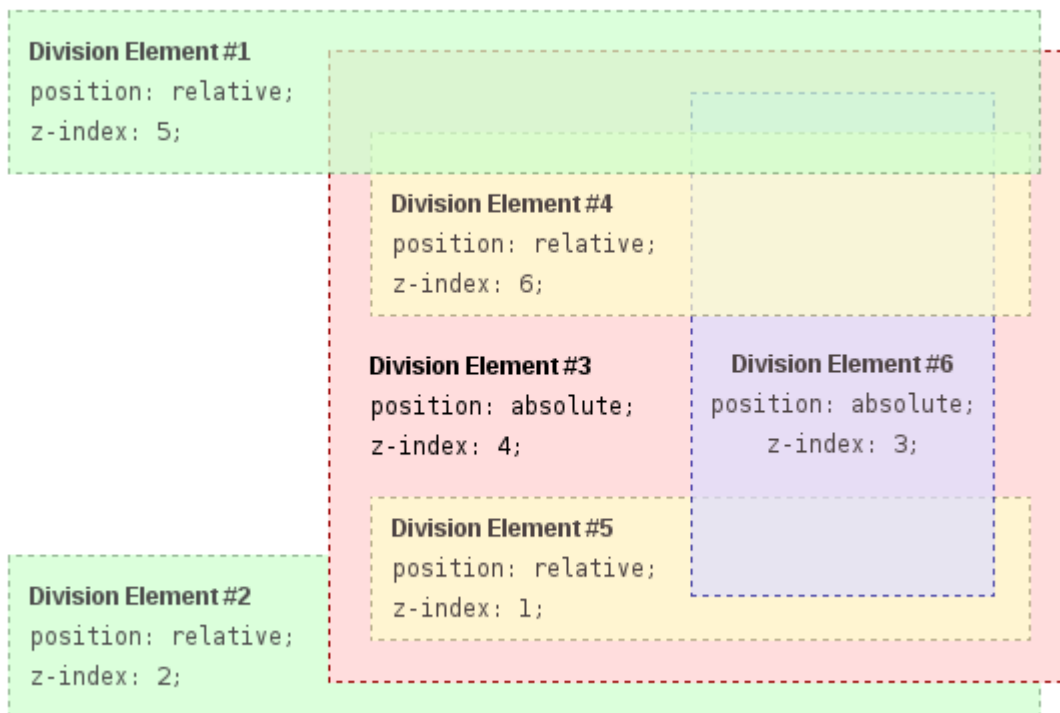
Read Single Element Shapes online: <https://riptutorial.com/css/topic/2862/single-element-shapes>

Chapter 50: Stacking Context

Examples

Stacking Context

In this example every positioned element creates its own stacking context, because of their positioning and z-index values. The hierarchy of stacking contexts is organized as follows:



- Root
 - DIV #1
 - DIV #2
 - DIV #3
 - DIV #4
 - DIV #5
 - DIV #6

It is important to note that DIV #4, DIV #5 and DIV #6 are children of DIV #3, so stacking of those elements is completely resolved within DIV #3. Once stacking and rendering within DIV #3 is completed, the whole DIV #3 element is passed for stacking in the root element with respect to its sibling's DIV.

HTML:

```
<div id="div1">  
  <h1>Division Element #1</h1>  
  <code>position: relative;<br/>
```

```

    z-index: 5;</code>
</div>
<div id="div2">
    <h1>Division Element #2</h1>
    <code>position: relative;<br/>
    z-index: 2;</code>
</div>
<div id="div3">
    <div id="div4">
        <h1>Division Element #4</h1>
        <code>position: relative;<br/>
        z-index: 6;</code>
    </div>
    <h1>Division Element #3</h1>
    <code>position: absolute;<br/>
    z-index: 4;</code>
    <div id="div5">
        <h1>Division Element #5</h1>
        <code>position: relative;<br/>
        z-index: 1;</code>
    </div>
    <div id="div6">
        <h1>Division Element #6</h1>
        <code>position: absolute;<br/>
        z-index: 3;</code>
    </div>
</div>

```

CSS:

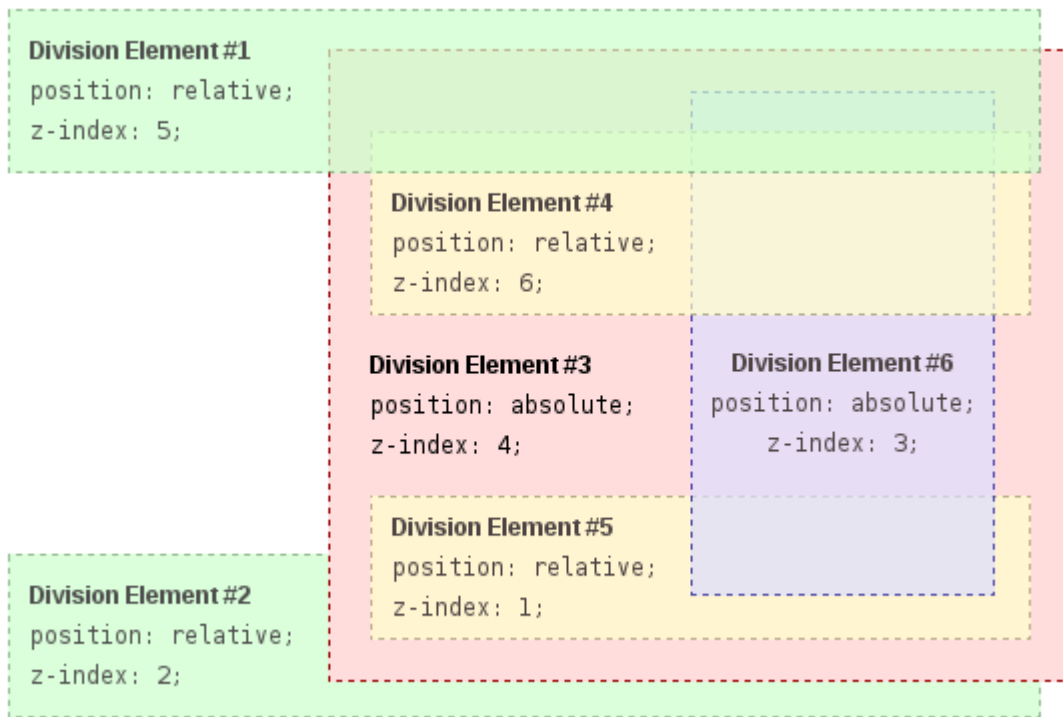
```

* {
    margin: 0;
}
html {
    padding: 20px;
    font: 12px/20px Arial, sans-serif;
}
div {
    opacity: 0.7;
    position: relative;
}
h1 {
    font: inherit;
    font-weight: bold;
}
#div1,
#div2 {
    border: 1px dashed #696;
    padding: 10px;
    background-color: #cfc;
}
#div1 {
    z-index: 5;
    margin-bottom: 190px;
}
#div2 {
    z-index: 2;
}
#div3 {
    z-index: 4;
}

```

```
    opacity: 1;
    position: absolute;
    top: 40px;
    left: 180px;
    width: 330px;
    border: 1px dashed #900;
    background-color: #fdd;
    padding: 40px 20px 20px;
}
#div4,
#div5 {
    border: 1px dashed #996;
    background-color: #ffc;
}
#div4 {
    z-index: 6;
    margin-bottom: 15px;
    padding: 25px 10px 5px;
}
#div5 {
    z-index: 1;
    margin-top: 15px;
    padding: 5px 10px;
}
#div6 {
    z-index: 3;
    position: absolute;
    top: 20px;
    left: 180px;
    width: 150px;
    height: 125px;
    border: 1px dashed #009;
    padding-top: 125px;
    background-color: #ddf;
    text-align: center;
}
```

Result:



Source: https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Positioning/Understanding_z_index/The_stacking_context.

Read Stacking Context online: <https://riptutorial.com/css/topic/5037/stacking-context>

Chapter 51: Structure and Formatting of a CSS Rule

Remarks

For ease of readability, keep all declarations indented one level from their selector, and the closing curly brace on its own line. Add a single space after selectors and colons, and always place a semicolon after the final declaration.

Good

```
p {  
    color: maroon;  
    font-size: 16px;  
}
```

Bad

```
p{  
    color: maroon;  
    font-size:16px }
```

One-Liner

If there are only one or two declarations, you *might* get away with this one. Not recommended for most cases. Always be consistent when possible.

```
p { color: maroon; font-size: 16px; }
```

Examples

Rules, Selectors, and Declaration Blocks

A CSS **rule** consists of a **selector** (e.g. `h1`) and **declaration block** (`{}`).

```
h1 {}
```

Property Lists

Some properties can take multiple values, collectively known as a **property list**.


```

/* Two values in this property list */
span {
    text-shadow: yellow 0 0 3px, green 4px 4px 10px;
}

/* Alternate Formatting */
span {
    text-shadow:
        yellow 0 0 3px,
        green 4px 4px 10px;
}

```

Multiple Selectors

When you group CSS selectors, you apply the same styles to several different elements without repeating the styles in your style sheet. Use a comma to separate multiple grouped selectors.

```
div, p { color: blue }
```

So the blue color applies to all `<div>` elements and all `<p>` elements. Without the comma only `<p>` elements that are a child of a `<div>` would be red.

This also applies to all types of selectors.

```
p, .blue, #first, div span{ color : blue }
```

This rule applies to:

- `<p>`
- elements of the `blue` class
- element with the ID `first`
- every `<span>` inside of a `<div>`

Read Structure and Formatting of a CSS Rule online:

<https://riptutorial.com/css/topic/4313/structure-and-formatting-of-a-css-rule>

Chapter 52: Tables

Syntax

- `table-layout`: *auto* | *fixed*;
- `border-collapse`: *separate* | *collapse*;
- `border-spacing`: `<length>` | `<length>` `<length>`;
- `empty-cells`: *show* | *hide*;
- `caption-side`: *top* | *bottom*;

Remarks

These properties apply to both `<table>` elements (*) and HTML elements displayed as `display: table` **OR** `display: inline-table`

(*) `<table>` elements are obviously natively styled by UA/browsers as `display: table`

HTML tables are semantically valid for tabular data. It is not recommended to use tables for layout. Instead, use CSS.

Examples

table-layout

The `table-layout` property changes the algorithm that is used for the layout of a table.

Below an example of two tables both set to `width: 150px`:

First name	Last name	Homeworld
Luke	Skywalker	Tatooine
Leia	Organa	Alderaan

First name	Last name	Homeworld
Luke	Skywalker	Tatooine
Leia	Organa	Alderaan

The table on the left has `table-layout: auto` while the one on the right has `table-layout: fixed`. The former is wider than the specified width (210px instead of 150px) but the contents fit. The latter takes the defined width of 150px, regardless if the contents overflow or not.

Value	Description
<i>auto</i>	This is the default value. It defines the layout of the table to be determined by the contents of its' cells.
<i>fixed</i>	This value sets the table layout to be determined by the width property provided to the table. If the content of a cell exceeds this width, the cell will not resize but

Value	Description
	instead, let the content overflow.

border-collapse

The `border-collapse` property determines if a tables' borders should be separated or merged.

Below an example of two tables with different values to the `border-collapse` property:

First name	Last name	Homeworld
Luke	Skywalker	Tatooine
Leia	Organa	Alderaan

First name	Last name	Homeworld
Luke	Skywalker	Tatooine
Leia	Organa	Alderaan

The table on the left has `border-collapse: separate` while the one on the right has `border-collapse: collapse`.

Value	Description
separate	This is the default value. It makes the borders of the table separate from each other.
collapse	This value sets the borders of the table to merge together, rather than being distinct.

border-spacing

The `border-spacing` property determines the spacing between cells. This has no effect unless `border-collapse` is set to `separate`.

Below an example of two tables with different values to the `border-spacing` property:

First name	Last name	Homeworld
Luke	Skywalker	Tatooine
Leia	Organa	Alderaan

First name	Last name	Homeworld
Luke	Skywalker	Tatooine
Leia	Organa	Alderaan

The table on the left has `border-spacing: 2px` (default) while the one on the right has `border-spacing: 8px`.

Value	Description
<code><length></code>	This is the default behavior, though the exact value can vary between browsers.
<code><length></code>	This syntax allows specifying separate horizontal and vertical values

Value	Description
<length>	respectively.

empty-cells

The `empty-cells` property determines if cells with no content should be displayed or not. This has no effect unless `border-collapse` is set to `separate`.

Below an example with two tables with different values set to the `empty-cells` property:

First name	Last name	Homeworld
Luke		Tatooine
Leia	Organa	

First name	Last name	Homeworld
Luke		Tatooine
Leia	Organa	

The table on the left has `empty-cells: show` while the one on the right has `empty-cells: hide`. The former does display the empty cells whereas the latter does not.

Value	Description
<i>show</i>	This is the default value. It shows cells even if they are empty.
hide	This value hides a cell altogether if there are no contents in the cell.

More Information:

- <https://www.w3.org/TR/CSS21/tables.html#empty-cells>
- <https://developer.mozilla.org/en-US/docs/Web/CSS/empty-cells>
- <http://codepen.io/SitePoint/pen/yfhtq>
- <https://css-tricks.com/almanac/properties/e/empty-cells/>

caption-side

The `caption-side` property determines the vertical positioning of the `<caption>` element within a table. This has no effect if such element does not exist.

Below an example with two tables with different values set to the `caption-side` property:

Star Wars figures		
First name	Last name	Homeworld
Luke	Skywalker	Tatooine
Leia	Organa	Alderaan

First name	Last name	Homeworld
Luke	Skywalker	Tatooine
Leia	Organa	Alderaan

| Star Wars figures | | |

The table on the left has `caption-side: top` while the one on the right has `caption-side: bottom`.

Value	Description
<i>top</i>	This is the default value. It places the caption above the table.
bottom	This value places the caption below the table.

Read Tables online: <https://riptutorial.com/css/topic/1074/tables>

Chapter 53: The Box Model

Syntax

- `box-sizing`: *parameter*;

Parameters

Parameter	Detail
<code>content-box</code>	Width and height of the element only includes content area.
<code>padding-box</code>	Width and height of the element includes content and padding.
<code>border-box</code>	Width and height of the element includes content, padding and border.
<code>initial</code>	Sets the box model to its default state.
<code>inherit</code>	Inherits the box model of the parent element.

Remarks

About padding-box

This value was only implemented by **Firefox** and thus should not be used. It was removed in Firefox version 50.0.

Examples

What is the Box Model?

The Edges

The browser creates a rectangle for each element in the HTML document. The Box Model describes how the padding, border, and margin are added to the content to create this rectangle.

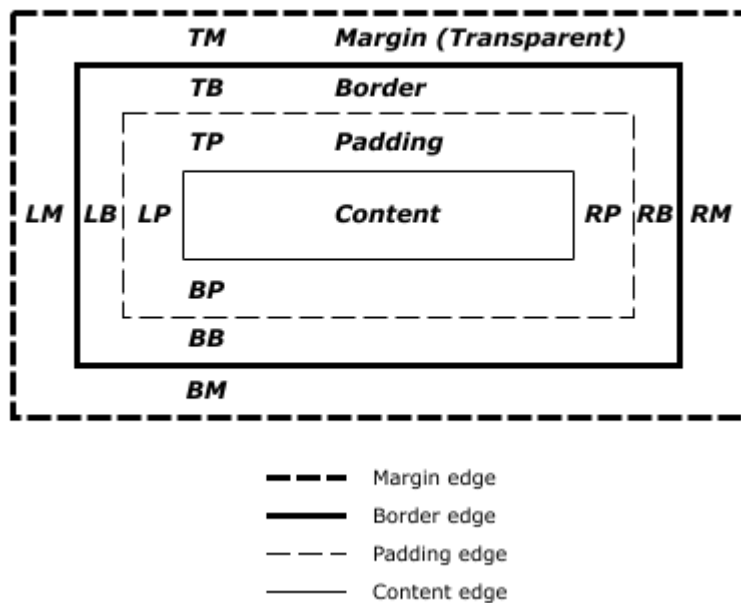


Diagram from [CSS2.2 Working Draft](#)

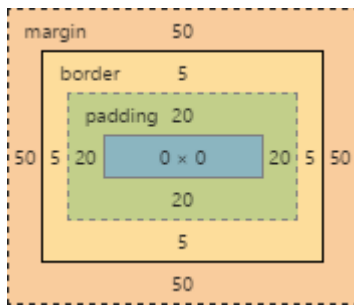
The perimeter of each of the four areas is called an *edge*. Each edge defines a *box*.

- The innermost rectangle is the **content box**. The width and height of this depends on the element's rendered content (text, images and any child elements it may have).
- Next is the **padding box**, as defined by the `padding` property. If there is no `padding` width defined, the padding edge is equal to the content edge.
- Then we have the **border box**, as defined by the `border` property. If there is no `border` width defined, the border edge is equal to the padding edge.
- The outermost rectangle is the **margin box**, as defined by the `margin` property. If there is no `margin` width defined, the margin edge is equal to the border edge.

Example

```
div {
  border: 5px solid red;
  margin: 50px;
  padding: 20px;
}
```

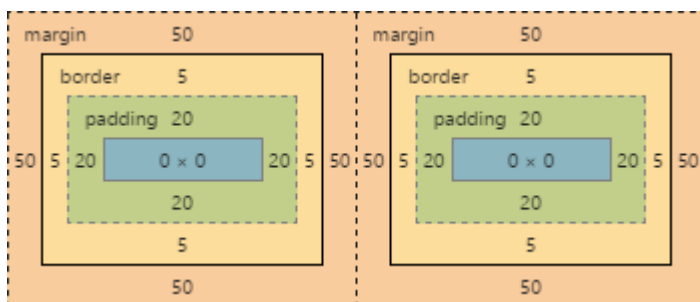
This CSS styles all `div` elements to have a top, right, bottom and left border of `5px` in width; a top, right, bottom and left margin of `50px`; and a top, right, bottom, and left padding of `20px`. Ignoring content, our generated box will look like this:



Screenshot of Google Chrome's Element Styles panel

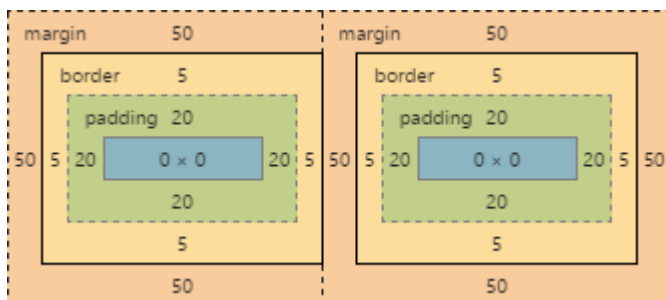
- As there is no content, the content region (the blue box in the middle) has no height or width (0px by 0px).
- The padding box by default is the same size as the content box, plus the 20px width on all four edges we're defining above with the `padding` property (40px by 40px).
- The border box is the same size as the padding box, plus the 5px width we're defining above with the `border` property (50px by 50px).
- Finally the margin box is the same size as the border box, plus the 50px width we're defining above with the `margin` property (giving our element a total size of 150px by 150px).

Now lets give our element a sibling with the same style. The browser looks at the Box Model of both elements to work out where in relation to the previous element's content the new element should be positioned:



The content of each of element is separated by a 150px gap, but the two elements' boxes touch each other.

If we then modify our first element to have no right margin, the right margin edge would be in the same position as the right border edge, and our two elements would now look like this:



box-sizing

The default box model (`content-box`) can be counter-intuitive, since the `width` / `height` for an

element will not represent its actual width or height on screen as soon as you start adding `padding` and `border` styles to the element.

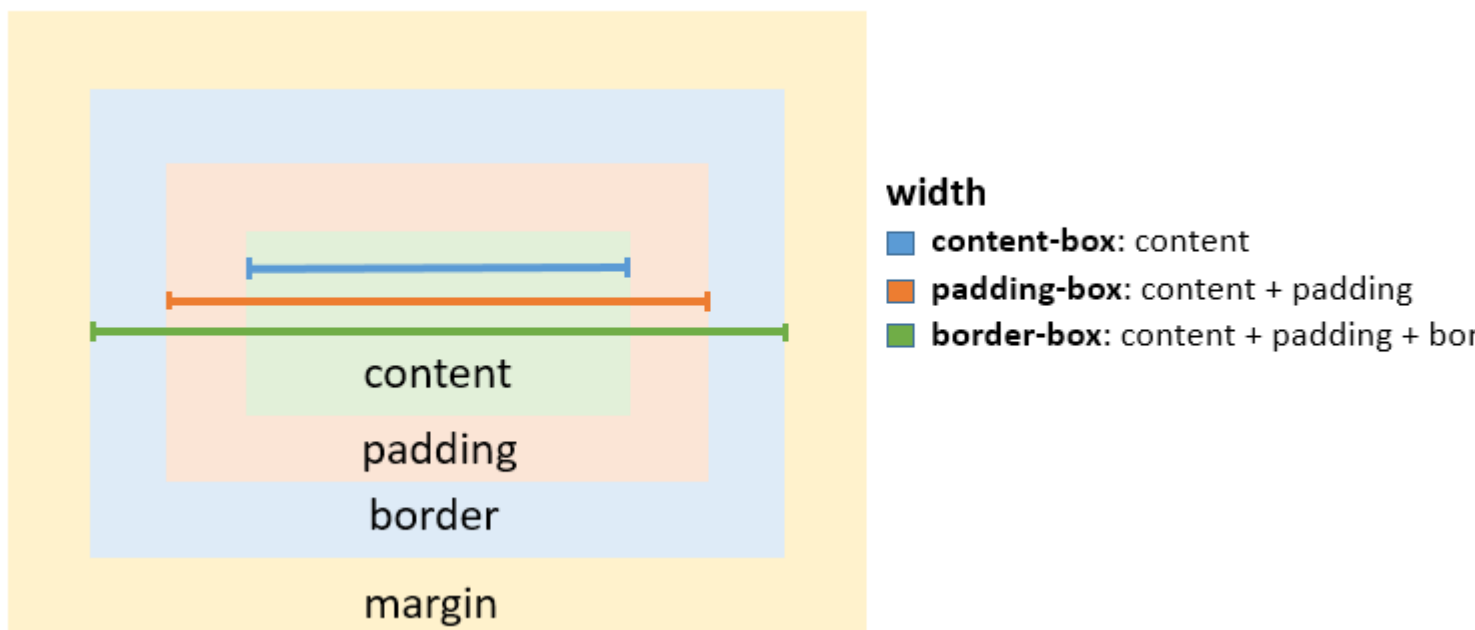
The following example demonstrates this potential issue with `content-box`:

```
textarea {  
  width: 100%;  
  padding: 3px;  
  box-sizing: content-box; /* default value */  
}
```

Since the padding will be added to the width of the textarea, the resulting element is a textarea that is wider than 100%.

Fortunately, CSS allows us to change the box model with the `box-sizing` property for an element. There are three different values for the property available:

- `content-box`: The common box model - width and height only includes the content, not the padding or border
- `padding-box`: Width and height includes the content and the padding, but not the border
- `border-box`: Width and height includes the content, the padding as well as the border



To solve the `textarea` problem above, you could just change the `box-sizing` property to `padding-box` or `border-box`. `border-box` is most commonly used.

```
textarea {  
  width: 100%;  
  padding: 3px;  
  box-sizing: border-box;  
}
```

To apply a specific box model to every element on the page, use the following snippet:

```
html {  
    box-sizing: border-box;  
}  
  
*, *:before, *:after {  
    box-sizing: inherit;  
}
```

In this coding `box-sizing: border-box;` is not directly applied to `*`, so you can easily overwrite this property on individual elements.

Read The Box Model online: <https://riptutorial.com/css/topic/646/the-box-model>

Chapter 54: Transitions

Syntax

- `transition: [transition-property] [transition-duration] [transition-timing-function] [transition-delay];`

Parameters

Parameter	Details
transition-property	The specific CSS property whose value change needs to be transitioned (or <code>all</code> , if all the transitionable properties need to be transitioned.
transition-duration	The duration (or period) in seconds (<code>s</code>) or milliseconds (<code>ms</code>) over which the transition must take place.
transition-timing-function	A function that describes how the intermediate values during the transition are calculated. Commonly used values are <code>ease</code> , <code>ease-in</code> , <code>ease-out</code> , <code>ease-in-out</code> , <code>linear</code> , <code>cubic-bezier()</code> , <code>steps()</code> . More information about the various timing functions can be found in the W3C specs .
transition-delay	The amount of time that must have elapsed before the transition can start. Can be specified in seconds (<code>s</code>) or milliseconds (<code>ms</code>)

Remarks

Some older browsers support only [vendor-prefixed](#) `transition` properties:

- `-webkit`: Chrome 25-, Safari 6-, Safari & Chrome for iOS 6.1-, Android 4.3- Browser, Blackberry Browser 7-, UC Browser 9.9- for Android.
- `-moz`: Firefox 15-.
- `-o`: Opera 11.5-, Opera Mobile 12-.

Example:

```
-webkit-transition: all 1s;  
-moz-transition: all 1s;  
-o-transition: all 1s;  
transition: all 1s;
```

Examples

Transition shorthand

CSS

```
div{
  width: 150px;
  height:150px;
  background-color: red;
  transition: background-color 1s;
}
div:hover{
  background-color: green;
}
```

HTML

```
<div></div>
```

This example will change the background color when the div is hovered the background-color change will last 1 second.

Transition (longhand)

CSS

```
div {
  height: 100px;
  width: 100px;
  border: 1px solid;
  transition-property: height, width;
  transition-duration: 1s, 500ms;
  transition-timing-function: linear;
  transition-delay: 0s, 1s;
}
div:hover {
  height: 200px;
  width: 200px;
}
```

HTML

```
<div></div>
```

- **transition-property:** Specifies the CSS properties the transition effect is for. In this case, the div will expand both horizontally and vertically when hovered.
- **transition-duration:** Specifies the length of time a transition takes to complete. In the above example, the height and width transitions will take 1 second and 500 milliseconds respectively.
- **transition-timing-function:** Specifies the speed curve of the transition effect. A *linear* value indicates the transition will have the same speed from start to finish.

- **transition-delay**: Specifies the amount of time needed to wait before the transition effect starts. In this case, the height will start transitioning immediately, whereas the width will wait 1 second.

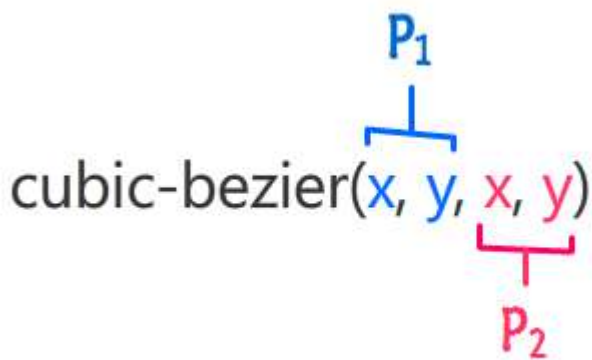
cubic-bezier

The `cubic-bezier` function is a transition timing function which is often used for custom and smooth transitions.

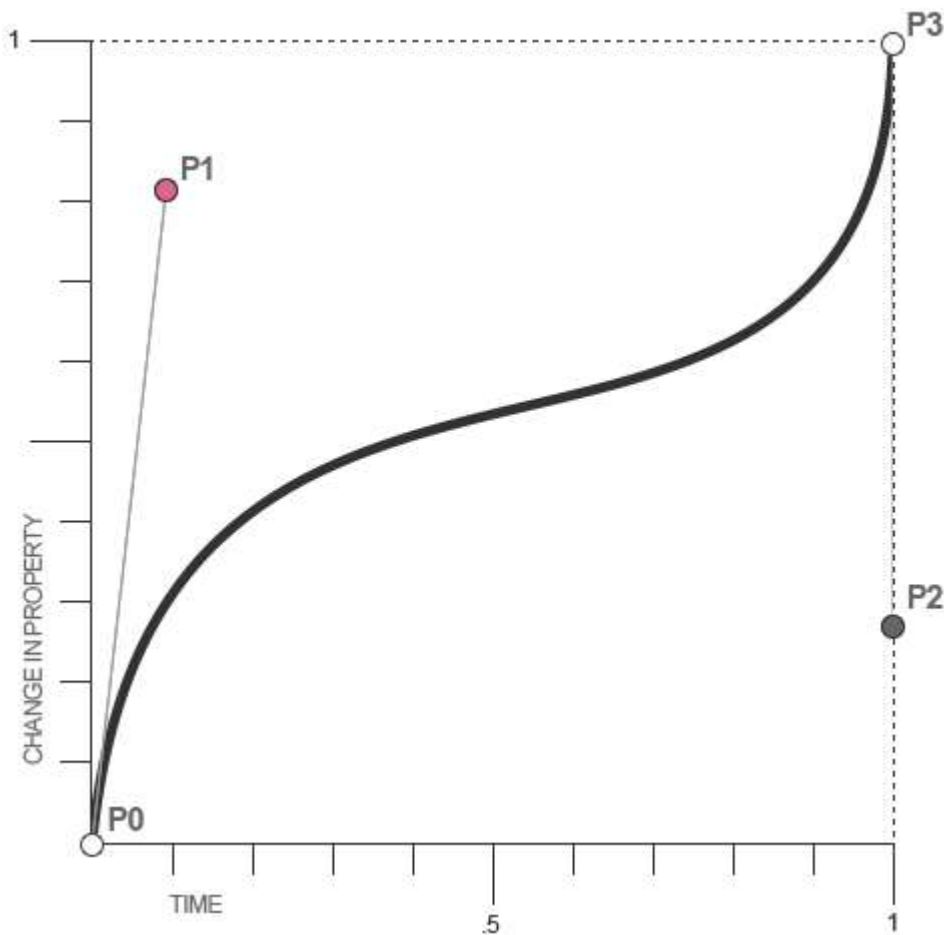
```
transition-timing-function: cubic-bezier(0.1, 0.7, 1.0, 0.1);
```

The function takes four parameters:

```
cubic-bezier(P1_x, P1_y, P2_x, P2_y)
```



These parameters will be mapped to points which are part of a [Bézier curve](#):



For CSS Bézier Curves, P0 and P3 are always in the same spot. P0 is at (0,0) and P3 is at (1,1), which means that the parameters passed to the cubic-bezier function can only be between 0 and 1.

If you pass parameters which aren't in this interval the function will default to a `linear` transition.

Since cubic-bezier is the most flexible transition in CSS, you can translate all other transition timing function to cubic-bezier functions:

```
linear: cubic-bezier(0,0,1,1)
```

```
ease-in: cubic-bezier(0.42, 0.0, 1.0, 1.0)
```

```
ease-out: cubic-bezier(0.0, 0.0, 0.58, 1.0)
```

```
ease-in-out: cubic-bezier(0.42, 0.0, 0.58, 1.0)
```

Read Transitions online: <https://riptutorial.com/css/topic/751/transitions>

Chapter 55: Typography

Syntax

- font: [*font-style*] [*font-variant*] [*font-weight*] *font-size* [*/line-height*] *font-family*;
- font-style: *font-style*
- font-variant: *font-variant*
- font-weight: *font-weight*;
- font-size: *font-size*;
- line-height: *line-height*;
- font-family: *font-family*;
- color: *color*;
- quotes: *none*|*string*|*initial*|*inherit*;
- font-stretch: *font-stretch*;
- text-align: *text-align*;
- text-indent: *length*|*initial*|*inherit*;
- text-overflow: *clip*|*ellipsis*|*string*|*initial*|*inherit*;
- text-transform: *none*|*capitalize*|*uppercase*|*lowercase*|*initial*|*inherit*;
- text-shadow: *h-shadow* *v-shadow* *blur-radius* *color*|*none*|*initial*|*inherit*;
- font-size-adjust: *number*|*none*|*initial*|*inherit*;
- font-stretch: *ultra-condensed*|*extra-condensed*|*condensed*|*semi-condensed*|*normal*|*semi-expanded*|*expanded*|*extra-expanded*|*ultra-expanded*|*initial*|*inherit*;
- hyphens: *none* | *manual* | *auto*;
- tab-size: *number*|*length*|*initial*|*inherit*;
- letter-spacing: *normal*|*length*|*initial*|*inherit*;
- word-spacing: *normal*|*length*|*initial*|*inherit*;

Parameters

Parameter	Details
<i>font-style</i>	<i>italics</i> or <i>oblique</i>
<i>font-variant</i>	<i>normal</i> or <i>small-caps</i>
<i>font-weight</i>	<i>normal</i> , <i>bold</i> or numeric from 100 to 900.
<i>font-size</i>	The font size given in %, px, em, or any other valid CSS measurement
<i>line-height</i>	The line height given in %, px, em, or any other valid CSS measurement
<i>font-family</i>	This is for defining the family's name.
<i>color</i>	Any valid CSS color representation , like <i>red</i> , <i>#00FF00</i> , <i>hsl(240, 100%, 50%)</i> etc.
<i>font-stretch</i>	Whether or not to use a condensed or expanded face from font. Valid values

Parameter	Details
	are normal, ultra-condensed, extra-condensed, condensed, semi-condensed, semi-expanded, expanded, extra-expanded Or ultra-expanded
<i>text-align</i>	start, end, left, right, center, justify, match-parent
<i>text-decoration</i>	none, underline, overline, line-through, initial, inherit;

Remarks

- The `text-shadow` property is not supported by versions of Internet Explorer less than 10.

Examples

Font Size

HTML:

```
<div id="element-one">Hello I am some text.</div>
<div id="element-two">Hello I am some smaller text.</div>
```

CSS:

```
#element-one {
    font-size: 30px;
}

#element-two {
    font-size: 10px;
}
```

The text inside `#element-one` will be 30px in size, while the text in `#element-two` will be 10px in size.

The Font Shorthand

With the syntax:

```
element {
    font: [font-style] [font-variant] [font-weight] [font-size/line-height] [font-family];
}
```

You can have all your font-related styles in one declaration with the `font` shorthand. Simply use the `font` property, and put your values in the correct order.

For example, to make all `p` elements bold with a font size of 20px and using Arial as the font family typically you would code it as follows:


```
p {
  font-weight: bold;
  font-size: 20px;
  font-family: Arial, sans-serif;
}
```

However with the font shorthand it can be condensed as follows:

```
p {
  font: bold 20px Arial, sans-serif;
}
```

Note: that since `font-style`, `font-variant`, `font-weight` and `line-height` are optional, the three of them are skipped in this example. It is important to note that using the shortcut **resets** the other attributes not given. Another important point is that the two necessary attributes for the font shortcut to work are `font-size` and `font-family`. If they are not both included the shortcut is ignored.

Initial value for each of the properties:

- `font-style: normal;`
- `font-variant: normal;`
- `font-weight: normal;`
- `font-stretch: normal;`
- `font-size: medium;`
- `line-height: normal;`
- `font-family` – depends on user agent

Font Stacks

```
font-family: 'Segoe UI', Tahoma, sans-serif;
```

The browser will attempt to apply the font face "Segoe UI" to the characters within the elements targeted by the above property. If this font is not available, or the font does not contain a glyph for the required character, the browser will fall back to Tahoma, and, if necessary, any sans-serif font on the user's computer. Note that any font names with more than one word such as "Segoe UI" need to have single or double quotes around them.

```
font-family: Consolas, 'Courier New', monospace;
```

The browser will attempt to apply the font face "Consolas" to the characters within the elements targeted by the above property. If this font is not available, or the font does not contain a glyph for the required character, the browser will fall back to "Courier New," and, if necessary, any monospace font on the user's computer.

Letter Spacing

```
h2 {
  /* adds a 1px space horizontally between each letter;
     also known as tracking */
  letter-spacing: 1px;
}
```

```
}
```

The `letter-spacing` property is used to specify the space between the characters in a text.

! `letter-spacing` also supports negative values:

```
p {  
  letter-spacing: -1px;  
}
```

Resources: <https://developer.mozilla.org/en-US/docs/Web/CSS/letter-spacing>

Text Transform

The `text-transform` property allows you to change the capitalization of text. Valid values are: `uppercase`, `capitalize`, `lowercase`, `initial`, `inherit`, and `none`

CSS:

```
.example1 {  
  text-transform: uppercase;  
}  
.example2 {  
  text-transform: capitalize;  
}  
.example3 {  
  text-transform: lowercase;  
}
```

HTML

```
<p class="example1">  
  all letters in uppercase <!-- "ALL LETTERS IN UPPERCASE" -->  
</p>  
<p class="example2">  
  all letters in capitalize <!-- "All Letters In Capitalize (Sentence Case)" -->  
</p>  
<p class="example3">  
  all letters in lowercase <!-- "all letters in lowercase" -->  
</p>
```

Text Indent

```
p {  
  text-indent: 50px;  
}
```

The `text-indent` property specifies how much horizontal space text should be moved before the beginning of the first line of the text content of an element.

Resources:

- [Indenting only the first line of text in a paragraph?](#)
- <https://www.w3.org/TR/CSS21/text.html#propdef-text-indent>
- <https://developer.mozilla.org/en-US/docs/Web/CSS/text-indent>

Text Decoration

The `text-decoration` property is used to set or remove decorations from text.

```
h1 { text-decoration: none; }
h2 { text-decoration: overline; }
h3 { text-decoration: line-through; }
h4 { text-decoration: underline; }
```

`text-decoration` can be used in combination with `text-decoration-style` and `text-decoration-color` as a shorthand property:

```
.title { text-decoration: underline dotted blue; }
```

This is a shorthand version of

```
.title {
  text-decoration-style: dotted;
  text-decoration-line: underline;
  text-decoration-color: blue;
}
```

It should be noted that the following properties are only supported in Firefox

- `text-decoration-color`
- `text-decoration-line`
- `text-decoration-style`
- `text-decoration-skip`

Text Overflow

The `text-overflow` property deals with how overflowed content should be signaled to users. In this example, the `ellipsis` represents clipped text.

```
.text {
  overflow: hidden;
  text-overflow: ellipsis;
}
```

Unfortunately, `text-overflow: ellipsis` only works on a single line of text. There is no way to support ellipsis on the last line in standard CSS, but it can be achieved with non-standard webkit-only implementation of flexboxes.

```
.giveMeEllipsis {
  overflow: hidden;
  text-overflow: ellipsis;
}
```

```
display: -webkit-box;
-webkit-box-orient: vertical;
-webkit-line-clamp: N; /* number of lines to show */
line-height: X;        /* fallback */
max-height: X*N;       /* fallback */
}
```

Example (open in Chrome or Safari):

<http://jsfiddle.net/csYjC/1131/>

Resources:

<https://www.w3.org/TR/2012/WD-css3-ui-20120117/#text-overflow0>

Word Spacing

The word-spacing property specifies the spacing behavior between tags and words.

Possible values

- a positive or negative *length* (using `em` `px` `vh` `cm` etc.) or *percentage* (using `%`)
- the keyword `normal` uses the font's default word spacing
- the keyword `inherit` takes the value from the parent element

CSS

```
.normal    { word-spacing: normal; }
.narrow    { word-spacing: -3px;  }
.extensive { word-spacing: 10px;  }
```

HTML

```
<p>
  <span class="normal">This is an example, showing the effect of "word-spacing".</span><br>
  <span class="narrow">This is an example, showing the effect of "word-spacing".</span><br>
  <span class="extensive">This is an example, showing the effect of "word-spacing".</span><br>
</p>
```

Online-Demo

[Try it yourself](#)

Further reading:

- [word-spacing – MDN](#)
- [word-spacing – w3.org](#)

Text Direction

```
div {
```

```

    direction: ltr; /* Default, text read from left-to-right */
}
.ex {
    direction: rtl; /* text read from right-to-left */
}
.horizontal-tb {
    writing-mode: horizontal-tb; /* Default, text read from left-to-right and top-to-bottom.
*/
}
.vertical-rtl {
    writing-mode: vertical-rl; /* text read from right-to-left and top-to-bottom */
}
.vertical-ltr {
    writing-mode: vertical-rl; /* text read from left-to-right and top to bottom */
}

```

The `direction` property is used to change the horizontal text direction of an element.

Syntax: `direction: ltr | rtl | initial | inherit;`

The `writing-mode` property changes the alignment of text so it can be read from top-to-bottom or from left-to-right, depending on the language.

Syntax: `direction: horizontal-tb | vertical-rl | vertical-lr;`

Font Variant

Attributes:

normal

Default attribute of fonts.

small-caps

Sets every letter to uppercase, **but** makes the lowercase letters(from original text) smaller in size than the letters that originally uppercase.

CSS:

```

.smallcaps{
    font-variant: small-caps;
}

```

HTML:

```

<p class="smallcaps">
    Documentation about CSS Fonts
    <br>
    aNd ExAmPlE
</p>

```

OUTPUT:

DOCUMENTATION ABOUT CSS FONTS
AND EXAMPLE

Note: The `font-variant` property is a shorthand for the properties: `font-variant-caps`, `font-variant-numeric`, `font-variant-alternates`, `font-variant-ligatures`, and `font-variant-east-asian`.

Quotes

The `quotes` property is used to customize the opening and closing quotation marks of the `<q>` tag.

```
q {  
    quotes: "«" "»";  
}
```

Text Shadow

To add shadows to text, use the `text-shadow` property. The syntax is as follows:

```
text-shadow: horizontal-offset vertical-offset blur color;
```

Shadow without blur radius

```
h1 {  
    text-shadow: 2px 2px #0000FF;  
}
```

This creates a blue shadow effect around a heading

Shadow with blur radius

To add a blur effect, add an option `blur radius` argument

```
h1 {  
    text-shadow: 2px 2px 10px #0000FF;  
}
```

Multiple Shadows

To give an element multiple shadows, separate them with commas

```
h1 {  
    text-shadow: 0 0 3px #FF0000, 0 0 5px #0000FF;  
}
```

Read Typography online: <https://riptutorial.com/css/topic/427/typography>

Chapter 56: Vertical Centering

Remarks

This is used when the element's dimensions (`width` and `height`) are not known or dynamic.

Prefer to use **Flexbox** over all other options as it is optimized for user interface design.

Examples

Centering with `display: table`

HTML:

```
<div class="wrapper">
  <div class="outer">
    <div class="inner">
      centered
    </div>
  </div>
</div>
```

CSS:

```
.wrapper {
  height: 600px;
  text-align: center;
}
.outer {
  display: table;
  height: 100%;
  width: 100%;
}
.outer .inner {
  display: table-cell;
  text-align: center;
  vertical-align: middle;
}
```

Centering with Transform

HTML:

```
<div class="wrapper">
  <div class="centered">
    centered
  </div>
</div>
```

CSS:

```
.wrapper {
  position: relative;
  height: 600px;
}
.centered {
  position: absolute;
  z-index: 999;
  transform: translate(-50%, -50%);
  top: 50%;
  left: 50%;
}
```

Centering with Flexbox

HTML:

```
<div class="container">
  <div class="child"></div>
</div>
```

CSS:

```
.container {
  height: 500px;
  width: 500px;
  display: flex;           // Use Flexbox
  align-items: center;     // This centers children vertically in the parent.
  justify-content: center; // This centers children horizontally.
  background: white;
}

.child {
  width: 100px;
  height: 100px;
  background: blue;
}
```

Centering Text with Line Height

HTML:

```
<div class="container">
  <span>vertically centered</span>
</div>
```

CSS:

```
.container{
  height: 50px;           /* set height */
  line-height: 50px;     /* set line-height equal to the height */
  vertical-align: middle; /* works without this rule, but it is good having it explicitly
set */
}
```


Note: This method will only vertically center a *single line of text*. It will not center block elements correctly and if the text breaks onto a new line, you will have two very tall lines of text.

Centering with Position: absolute

HTML:

```
<div class="wrapper">
  
</div>
```

CSS:

```
.wrapper{
  position:relative;
  height: 600px;
}
.wrapper img {
  position: absolute;
  top: 0;
  left: 0;
  right: 0;
  bottom: 0;
  margin: auto;
}
```

If you want to center other than images, then you must give height and width to that element.

HTML:

```
<div class="wrapper">
  <div class="child">
    make me center
  </div>
</div>
```

CSS:

```
.wrapper{
  position:relative;
  height: 600px;
}
.wrapper .child {
  position: absolute;
  top: 0;
  left: 0;
  right: 0;
  bottom: 0;
  margin: auto;
  width: 200px;
  height: 30px;
  border: 1px solid #f00;
}
```

Centering with pseudo element

HTML:

```
<div class="wrapper">
  <div class="content"></div>
</div>
```

CSS:

```
.wrapper{
  min-height: 600px;
}

.wrapper:before{
  content: "";
  display: inline-block;
  height: 100%;
  vertical-align: middle;
}

.content {
  display: inline-block;
  height: 80px;
  vertical-align: middle;
}
```

This method is best used in cases where you have a varied-height `.content` centered inside `.wrapper`; and you want `.wrapper`'s height to expand when `.content`'s height exceed `.wrapper`'s min-height.

Read Vertical Centering online: <https://riptutorial.com/css/topic/5070/vertical-centering>

Credits

S. No	Chapters	Contributors
1	Getting started with CSS	adamboro , animuson , Ashwin Ramaswami , awe , Boysenb3rry , Chris , Community , csx.cc , darrylyeo , FelipeAls , Gabriel R. , Garconis , Gerardas , GoatsWearHats , G-Wiz , Harish Gyanani , Heri Hehe Setiawan , J Atkin , Jmh2013 , joe_young , Jose Gomez , Just a student , Lambda Ninja , Marjorie Pickard , Nathan Arthur , patelarpan , RamenChef , Rocket Risa , Saroj Sasmal , ScientiaEtVeritas , selvassn , Sverri M. Olsen , Teo Dragovic , Todd , TylerH , Vivek Ghaisas , Xinyang Li , ZaneDickens
2	2D Transforms	Charlie H , Christiaan Maks , Harry , Hors Sujet , John Slegers , Luke Taylor , Madalina Taina , mnoronha , PaMaDo , Praveen Kumar , RamenChef , web-tiki , zer00ne
3	3D Transforms	Harry , Luka Kerr , Madalina Taina , mnoronha , Mr. Meeseeks , Nhan , RamenChef , web-tiki
4	Animations	Aeolingamenfel , apaul , Dex Star , Jasmin Solanki , Nathan Arthur , RamenChef , Richard Hamilton , TylerH
5	Backgrounds	4444 , Ahmad Alfy , animuson , Asim K T , Ben Rhys-Lewis , Boris , CalvT , cdm , Charlie H , CocoaBean , Dan Devine , Dan Eastwell , Daniel G. Blázquez , Daniel Stradowski , Darthstroke , designcise , Devid Farinelli , dippas , fcalderan , FelipeAls , Goose , Horst Jahns , Hynes , Jack , Jacob Gray , James Taylor , John Slegers , Jon Chan , Jonathan Zúñiga , Kevin Montrose , Louis St-Amour , Madalina Taina , Maximillian Laumeister , Michael Moriarty , Mr. Alien , mtb , Nate , Nathan Arthur , Nhan , Persijn , Praveen Kumar , RamenChef , Richard Hamilton , ScientiaEtVeritas , Sergey Denisov , Shaggy , Sourav Ghosh , Stewartside , Stratboy , think123 , Timothy , Trevor Clarke , TylerH , Zac , Zeta , Zze
6	Block Formatting Contexts	Madalina Taina , Milan Laslop , RamenChef
7	Border	andreas , Cassidy Williams , doctorsherlock , FelipeAls , Gnietschow , Harry , jaredsk , Madalina Taina , Nobal Mohan , RamenChef , ScientiaEtVeritas , Trevor Clarke
8	box-shadow	Hristo , Madalina Taina , RamenChef
9	Browser Support & Prefixes	Andrew , animuson , Braiam , Nhan , Obsidian , RamenChef , ScientiaEtVeritas , Shaggy , TylerH

10	Cascading and Specificity	amflare , Arjan Einbu , brandaemon , DarkAjax , dippas , dmkerr , geeksal , Grant Palin , G-Wiz , James Donnelly , jehna1 , John Slegers , kingcobra1986 , Matas Vaitkevicius , Nathan Arthur , Nhan , Ortomala Lokni , RamenChef , ScientiaEtVeritas , Sunnyok , Ze Rubeus
11	Centering	abaracedo , Alex Morales , Alohci , andreas , animuson , apaul , Christiaan Maks , Daniel Käfer , Devid Farinelli , Diego V , dippas , Eliran Malka , Emanuele Parisio , Euan Williams , F. Müller , Farzad YZ , Felix A J , geek1011 , insertusernamehere , JedaiCoder , jehna1 , JHS , John Slegers , Jonathan Argentiero , Kilian Stinson , Kyle Ratliff , Lambda Ninja , Lucia Bentivoglio , Luke Taylor , Madalina Taina , maho , Maxouhell , Michael_B , Mifeet , Mod Proxy , Mohammad Usman , Nathan Arthur , Nhan , o.v. , Ortomala Lokni , paaacman , Paul Kozlovitch , Praveen Kumar , Sandeep Tuniki , ScientiaEtVeritas , Sergej , Siavas , smonff , Someone , Stewartside , Sunnyok , Taylor , TylerH , web-tiki , Ze Rubeus , zeel
12	Clipping and Masking	Andre Lopes , Harry , RamenChef , ScientiaEtVeritas , web-tiki
13	Colors	andreas , animuson , Arjan Einbu , Brett DeWoody , Community , cuervoo , darrylyeo , designcise , H. Pauwelyn , Jasmin Solanki , John Slegers , Kuhan , Marc , Michael Moriarty , Miro , Nathan Arthur , niyasc , RamenChef , Richard Hamilton , ScientiaEtVeritas , SeinopSys , Stewartside , user007 , Wolfgang , X-27
14	Columns	Brett DeWoody , Madalina Taina , RamenChef
15	Comments	animuson , bdkopen , coderfin , Madalina Taina , Nick
16	Counters	Harry , RamenChef
17	CSS design patterns	John Slegers
18	CSS Image Sprites	Elegant.Scripting , Jmh2013 , RamenChef , Ted Goas , Ulrich Schwarz
19	CSS Object Model (CSSOM)	Alohci , animuson , feeela , Paul Sweatte , RamenChef , rishabh dev
20	Cursor Styling	cone56 , Madalina Taina , ScientiaEtVeritas , Squazz
21	Custom Properties (Variables)	animuson , Brett DeWoody , Community , Daniel Käfer , Muthu Kumaran , Obsidian , RamenChef , RedRiderX , TylerH
22	Feature Queries	Andrew Myers , RamenChef
23	Filter Property	Jeffery Tang , Nathan , RamenChef

24	Flexible Box Layout (Flexbox)	Ahmad Alfy , Asim K T , FelipeAls , fzzylogic , James Donnelly , Jef , Lambda Ninja , Marc-Antoine Leclerc , Nathan Arthur , Nhan , Ori Marash , RamenChef , Randy , Squazz , takeradi , Ted Goas , Timothy Morris , TylerH
25	Floats	demonofthemist , FelipeAls , Madalina Taina , Nathan Arthur , RamenChef , vishak
26	Fragmentation	animuson , dodopok , Madalina Taina , Milan Laslop , RamenChef
27	Functions	animuson , Brett DeWoody , cone56 , dodopok , H. Pauwelyn , haim770 , jaredsk , khawarPK , Kobi , RamenChef , SeinopSys , TheGenie OfTruth , TylerH
28	Grid	Chris Spittles , FelipeAls , Jonathan Zúñiga , Mike McCaughan , Nhan , Praveen Kumar , RamenChef , Sebastian Zartner
29	Inheritance	Chris , mnoronha , RamenChef
30	Inline-Block Layout	Marten Koetsier , RamenChef
31	Internet Explorer Hacks	Aeolingamenfel , animuson , Elizaveta Revyakina , feeela , John Slegers , LiLacTac , Praveen Kumar , RamenChef , ScientiaEtVeritas , Timothy Miller , TylerH
32	Layout Control	Arjun Iv , Chathuranga Jayanath , Chris , Jmh2013 , Kevin Katzke , Kurtis Beavers , Madalina Taina , mnoronha , Niek Brouwer , RamenChef , Sander Koedood , ScientiaEtVeritas , SeinopSys
33	Length Units	4dgaaurav , A B , animuson , Epodax , geeksal , J F , Marc , Milche Patern , Ortomala Lokni , RamenChef , Richard Hamilton , rmondesilva , Robert Koritnik , Robotnicka , ScientiaEtVeritas , StefanBob , Stewartside , Thomas Altmann , Toby , user2622348 , vladdobra , Zakaria Acharki , zer00ne
34	List Styles	animuson , Madalina Taina , Marten Koetsier , RamenChef , Ted Goas
35	Margins	Arjan Einbu , cdm , Chris Spittles , Community , J F , Madalina Taina , Mr_Green , Nathan Arthur , RamenChef , rejnev , Sun Qingyao , Sunnyok , Tot Zam , Trevor Clarke
36	Media Queries	amflare , Chathuranga Jayanath , darrylyeo , Demeter Dimitri , dodopok , James Donnelly , Jmh2013 , joe_young , joejoe31b , John Slegers , Matas Vaitkevicius , Mattia Astorino , Maximillian Laumeister , Nathan Arthur , Praveen Kumar , RamenChef , ScientiaEtVeritas , srikarg , Teo Dragovic , Viktor
37	Multiple columns	bipon , Sebastian Zartner

38	Normalizing Browser Styles	andre mcgruder , Confused One , Grant Palin , MMachinegun , mnoronha , RamenChef , SeinopSys
39	Object Fit and Placement	4444 , Miles , RamenChef
40	Opacity	Andrew , animuson , Hillel Tech , Madalina Taina , RamenChef
41	Outlines	Arif , Casey , Chathuranga Jayanath , Daniel Nugent , FelipeAls , Madalina Taina , RamenChef , ScientiaEtVeritas
42	Overflow	Andrew , bdkopen , jgh , Madalina Taina , Miles , Qaz , RamenChef , ScientiaEtVeritas , Ted Goas , Zac
43	Padding	Andy G , CalvT , Felix A J , Madalina Taina , Mehdi Dehghani , Nathan , Paul Sweatte , pixelbandito , RamenChef , StefanBob , Will DiFruscio
44	Performance	mnoronha , RamenChef , TrungDQ
45	Positioning	Abhishek Singh , Alohci , animuson , Blunderfest , CalvT , Cassidy Williams , Chetan Joshi , GingerPlusPlus , Jacob Gray , Lambda Ninja , Madalina Taina , Matthew Beckman , Mattia Astorino , Rahul Nanwani , RamenChef , Sourav Ghosh , Stephen Leppik , Theodore K. , TylerH
46	Pseudo-Elements	4dgaaurav , ankit , Chris , Community , dippas , dmnsgn , geek1011 , geeksal , Gofilord , kevin vd , Marcatectura , Milche Patern , Nathan , Pat , Praveen Kumar , RamenChef , ScientiaEtVeritas , Shaggy , Stewartside , Sunnyok
47	Selectors	75th Trombone , A.J , Aaron , abaracedo , Ahmad Alfy , Alex Filatov , amflare , Anil , animuson , Araknid , Arjan Einbu , Ashwin Ramaswami , BoltClock , Cerbrus , Charlie H , Chris , Chris Nager , Clinton Yeboah , Community , CPHPython , darrylyeo , Dave Everitt , David Fullerton , Demeter Dimitri , designcise , Devid Farinelli , Devon Bernard , Dinidu , dippas , Erenor Paz , Felix Edelmann , Felix Schütz , flyingfisch , Forty , fracz , Frits , gandreadis , geeksal , George Bailey , George Grigorita , H. Pauwelyn , HansCz , henry , Hugo Buff , Hynes , J Atkin , J F , Jacob Gray , James Donnelly , James Taylor , Jasha , jehna1 , Jmh2013 , joejoe31b , Joël Bonet Rodríguez , John Slegers , Kurtis Beavers , Madalina Taina , Marc , Mark Perera , Matas Vaitkevicius , Matsemann , Michael_B , Milan Laslop , Naeem Shaikh , Nathan Arthur , Nick , Ortomala Lokni , Persijn , Praveen Kumar , RamenChef , rdans , Richard Hamilton , Rion Williams , Robert Koritnik , RockPaperLizard , RoToRa , Sbats , ScientiaEtVeritas , Shaggy , Siavas , Stewartside , sudo bangbang , Sumner Evans , Sunnyok , ThatWeirdo , theB , Thomas Gerot ,

		TylerH , xpy , Yury Fedorov , Zac , Zaffy , Zaz , Ze Rubeus , Zze
48	Shapes for Floats	animuson , Harry , RamenChef , ScientiaEtVeritas
49	Single Element Shapes	andreas , animuson , Brett DeWoody , Chiller , J F , Nick , RamenChef , ScientiaEtVeritas , TylerH
50	Stacking Context	Nemanja Trifunovic , RamenChef
51	Structure and Formatting of a CSS Rule	Alon Eitan , darrylyeo , Marjorie Pickard
52	Tables	Casper Spruit , Chris , FelipeAls , JHS , Madalina Taina
53	The Box Model	Arjan Einbu , Ben Rhys-Lewis , Benolot , BiscuitBaker , FelipeAls , James Donnelly , Lambda Ninja , Nathan Arthur , Ortomala Lokni , RamenChef , ScientiaEtVeritas , Sergej , V-Kopio
54	Transitions	Christiaan Maks , dippas , Harry , Praveen Kumar , RamenChef , Richard Hamilton , ScientiaEtVeritas , Sergey Denisov , web-tiki
55	Typography	Alex Morales , Alohci , andreas , Arjan Einbu , ChaoticTwist , Evgeny , Felix A J , Goulven , Hynes , insertusernamehere , James Donnelly , joe_young , Jon Chan , Madalina Taina , Michael Moriarty , Nathan Arthur , Nhan , Praveen Kumar , RamenChef , Richard Hamilton , rmondesilva , Ryan , Sourav Ghosh , Suhaib Janjua , Ted Goas , Toby , ToniB , Trevor Clarke , user2622348 , Vlusion , Volker E.
56	Vertical Centering	animuson , AVAVT , bocanegra , Chiller , Chris , jaredsk , leo_ap , mmativ , patelarpan , Phil , Praveen Kumar , RamenChef