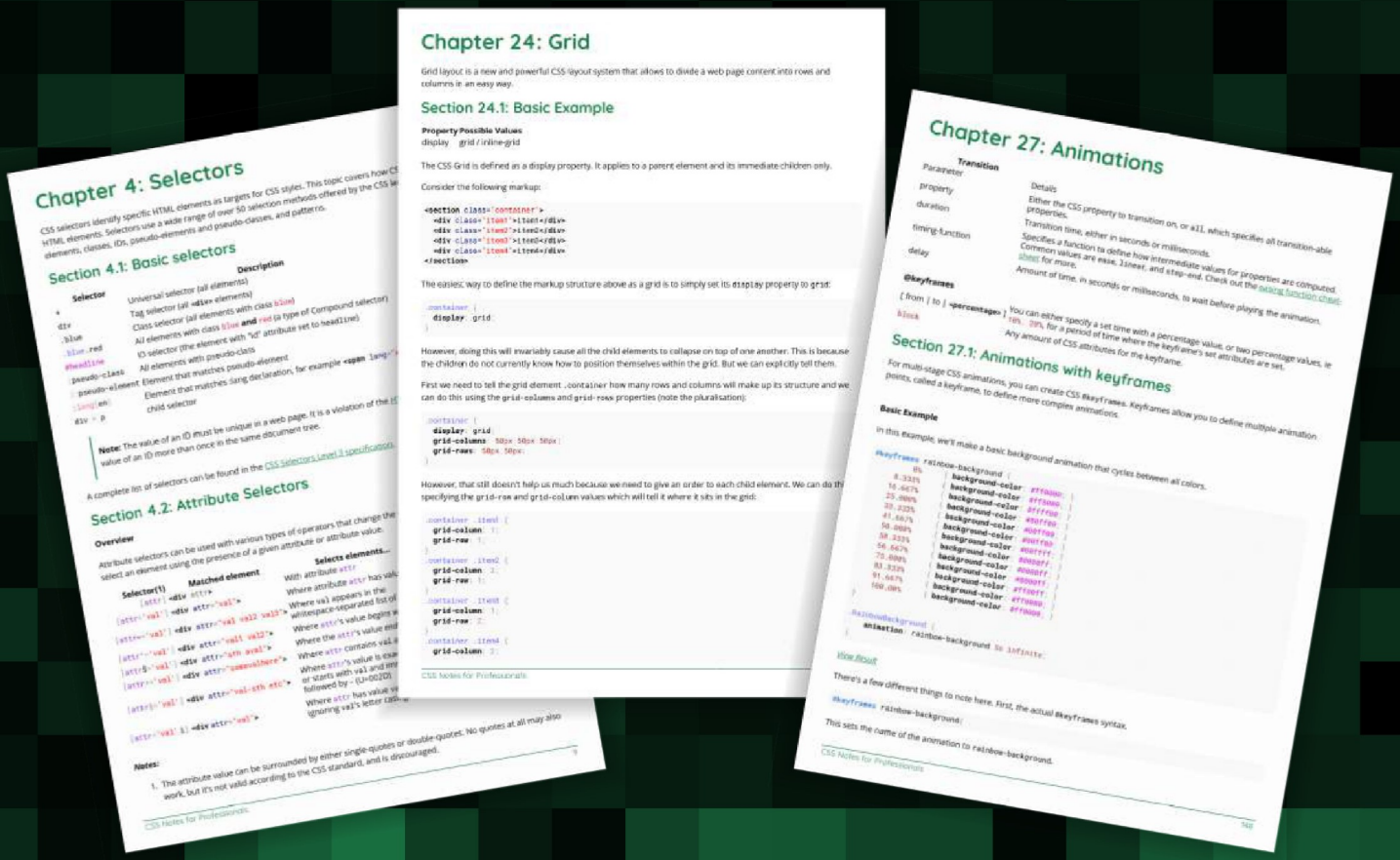
CSS

Notes for Professionals



200+ pages

of professional hints and tricks

Contents

About

Please feel free to share this PDF with anyone for free,
latest version of this book can be downloaded from:
https://goalkicker.com/CSSBook

Please send feedback and corrections to web@petercv.com

Chapter 1: Getting started with CSS

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

Section 1.1: 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.

Section 1.2: 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>
```

Section 1.3: 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);
```

Section 1.4: 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.

Section 1.5: 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

Section 1.6: 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;
}
```

Chapter 2: Structure and Formatting of a CSS Rule

Section 2.1: 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;
}
```

Section 2.2: 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>`

Section 2.3: Rules, Selectors, and Declaration Blocks

A CSS **rule** consists of a **selector** (e.g. `h1`) and **declaration block** (`{}`).

```
h1 {}
```

Chapter 3: Comments

Section 3.1: Single Line

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

Section 3.2: Multiple Line

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

Chapter 4: Selectors

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.

Section 4.1: Basic selectors

Selector	Description
`*`	Universal selector (all elements)
`div`	Tag selector (all `<div>` elements)
`.blue`	Class selector (all elements with class `blue`)
`.blue.red`	All elements with class `blue` **and** `red` (a type of Compound selector)
`#headline`	ID selector (the element with "id" attribute set to `headline`)
`:pseudo-class`	All elements with pseudo-class
`::pseudo-element`	Element that matches pseudo-element
`:lang(en)`	Element that matches :lang declaration, for example `<span lang="en">`
`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.

Section 4.2: 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 *begins* with `val`	3
`[attr$='val']`	`<div attr="sth aval">`	Where the `attr`'s value *ends* 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="barfooo123">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.

Section 4.3: Combinators

Overview

Selector	Description
`div span`	Descendant selector (all **`<span>`**s that are descendants of a **`<div>`**)
`div > span`	Child selector (all **`<span>`**s that are a direct child of a **`<div>`**)
`a ~ span`	General Sibling selector (all **`<span>`**s that are siblings after an **`<a>`**)
`a + span`	Adjacent Sibling selector (all **`<span>`**s that are immediately after an **`<a>`**)

> **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 immediate 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.

Section 4.4: 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
`:active`	Applies to any element being activated (i.e. clicked) by the user.
`:any`	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.
`:target`	Selects the current active #news element (clicked on a URL containing that anchor name)
`:checked`	Applies to radio, checkbox, or option elements that are checked or toggled into an "on" state.
`:default`	Represents any user interface element that is the default among a group of similar elements.
`:disabled`	Applies to any UI element which is in a disabled state.
`:empty`	Applies to any element which has no children.
`:enabled`	Applies to any UI element which is in an enabled state.
`:first`	Used in conjunction with the `@page` rule, this selects the first page in a printed document.
`:first-child`	Represents any element that is the first child element of its parent.
`:first-of-type`	Applies when an element is the first of the selected element type inside its parent. This may or may not be the first-child.
`:focus`	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.
`:focus-within`	Can be used to highlight a whole section when one element inside it is focused. It matches any element that the :focus pseudo-class matches or that has a descendant focused.
`:full-screen`	Applies to any element displayed in full-screen mode. It selects the whole stack of elements and not just the top level element.
`:hover`	Applies to any element being hovered by the user's pointing device, but not activated.
`:indeterminate`	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.
`:in-range`	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.
`:invalid`	Applies to **`<input>`** elements whose values are invalid according to the type specified in the `type=` attribute.
`:lang`	Applies to any element who's wrapping **`<body>`** element has a properly designated `lang=` attribute. For the pseudo-class to be valid, it must contain a valid two or three letter language code.
`:last-child`	Represents any element that is the last child element of its parent.
`:last-of-type`	Applies when an element is the last of the selected element type inside its parent. This may or may not be the last-child.

`:left`	Used in conjunction with the `@page` rule, this selects all the left pages in a printed document.
`:link`	Applies to any links which haven't been visited by the user.
`:not()`	Applies to all elements which **do not** match the value passed to (`:not(p)` or `:not(.class-name)` for example. It must have a value to be valid and it can only contain one selector. However, you can chain multiple `:not` selectors together.
`:nth-child`	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 `odd` or `even`.
`:nth-of-type`	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 `odd` or `even`.
`:only-child`	The `:only-child` CSS pseudo-class represents any element which is the only child of its parent. This is the same as `:first-child:last-child` or `:nth-child(1):nth-last-child(1)`, but with a lower specificity.
`:optional`	The `:optional` 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.
`:out-of-range`	The `:out-of-range` 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.
`:placeholder-shown`	**Experimental.** Applies to any form element currently displaying placeholder text.
`:read-only`	Applies to any element which is not editable by the user.
`:read-write`	Applies to any element that is editable by a user, such as `<input>` elements.
`:right`	Used in conjunction with the `@page` rule, this selects all the right pages in a printed document.
`:root`	matches the root element of a tree representing the document.
`:scope`	CSS pseudo-class matches the elements that are a reference point for selectors to match against.
`:target`	Selects the current active #news element (clicked on a URL containing that anchor name)
`:visited`	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.

Section 4.5: 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)`	✓	✓	✓							
`:nth-child(odd)`	✓		✓		✓		✓		✓	
`:nth-child(even)`		✓		✓		✓		✓		✓
`:last-child`										✓
`:nth-last-child(3)`							✓			

Section 4.6: Class Name Selectors

The class name selector select 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 with 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 names and child elements with `.important` class names.

Section 4.7: 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

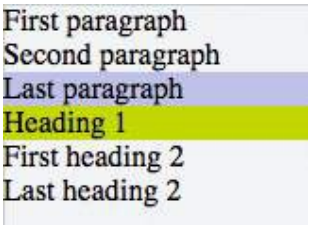
```
#element { ... } /* High specificity will override many selectors */

[id="element"] { ... } /* Low specificity, can be overridden easily */
```

Section 4.8: 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>
```



jsFiddle

Section 4.9: 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]

Section 4.10: 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>
```

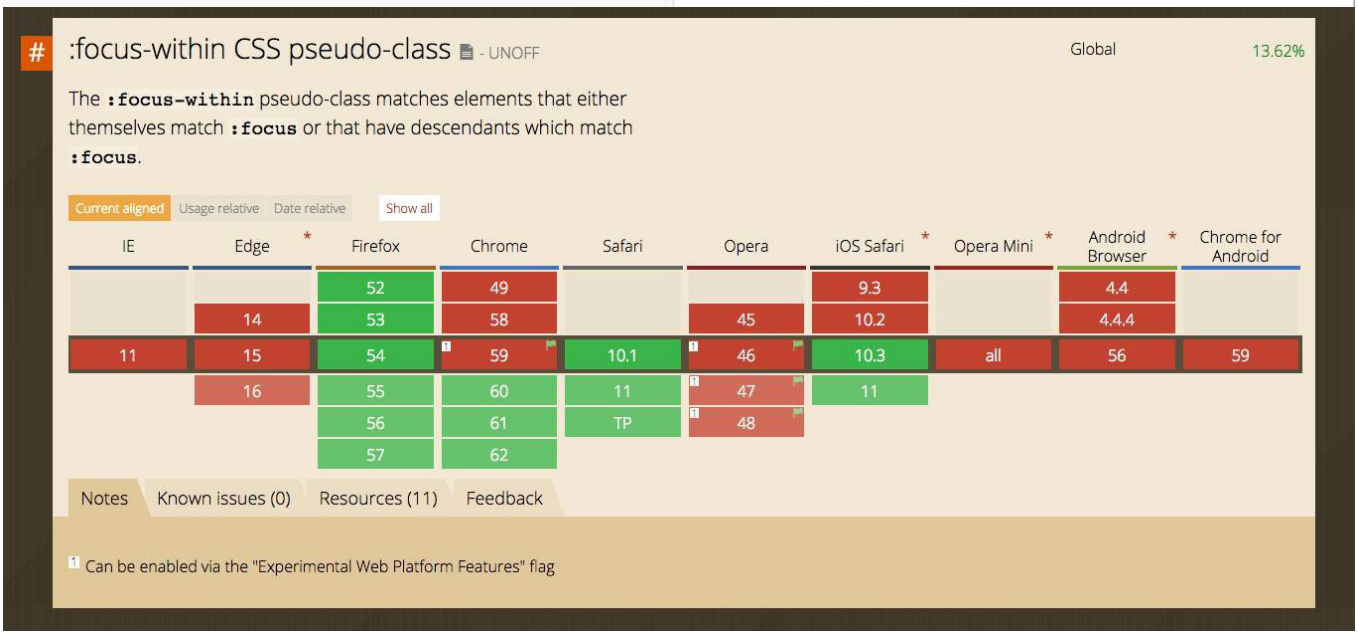
CSS:

```
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 .



Section 4.11: 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.

Section 4.12: 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

Section 4.13: How to style a Range input

HTML

```
<input type="range"></input>
```

CSS

Effect	Pseudo Selector
Thumb	`input[type=range]::-webkit-slider-thumb`, `input[type=range]::-moz-range-thumb`, `input[type=range]::-ms-thumb`
Track	`input[type=range]::-webkit-slider-runnable-track`, `input[type=range]::-moz-range-track`, `input[type=range]::-ms-track`
OnFocus	`input[type=range]:focus`
Lower part of the track	`input[type=range]::-moz-range-progress`, `input[type=range]::-ms-fill-lower` (not possible in WebKit browsers currently - JS needed)

Section 4.14: 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

Chapter 5: Backgrounds

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.

Section 5.1: 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

Section 5.2: 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 `to top`, `to bottom`, `to right` or `to left`; or an angle as `0deg`, `90deg`... . The angle starts from to top 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, `yellow 10%`, `rgba(0,0,0,.5) 40px`, `#fff 100%`...

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

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 deg, grad, rad, or turn.
`to left`	Direction of gradient, default is `to bottom`. Syntax: `to [y-axis(top OR bottom)] [x-axis(left OR right)]` ie `to top right`
`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

Section 5.3: 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
`url('/path/to/image.jpg')`	Specify background image's path(s) or an image resource specified with data URI schema (apostrophes can be omitted), separate multiples by comma
`none`	No background image
`initial`	Default value
`inherit`	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;
```

Section 5.4: Background Shorthand

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

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

Section 5.5: Background Size

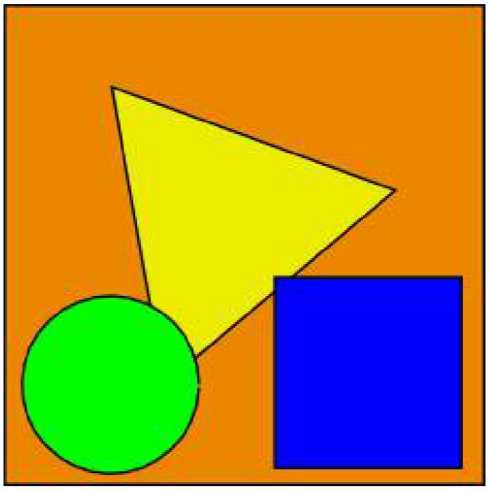
General overview

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 and horizontal direction. If the property is missing, its 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;
```

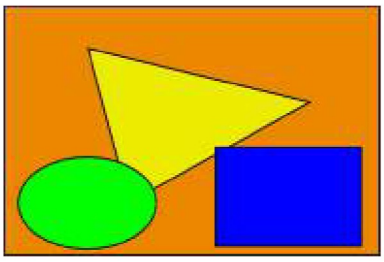
So if we started with the following picture (which has the mentioned size of 256 px × 256 px),



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 previos 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`

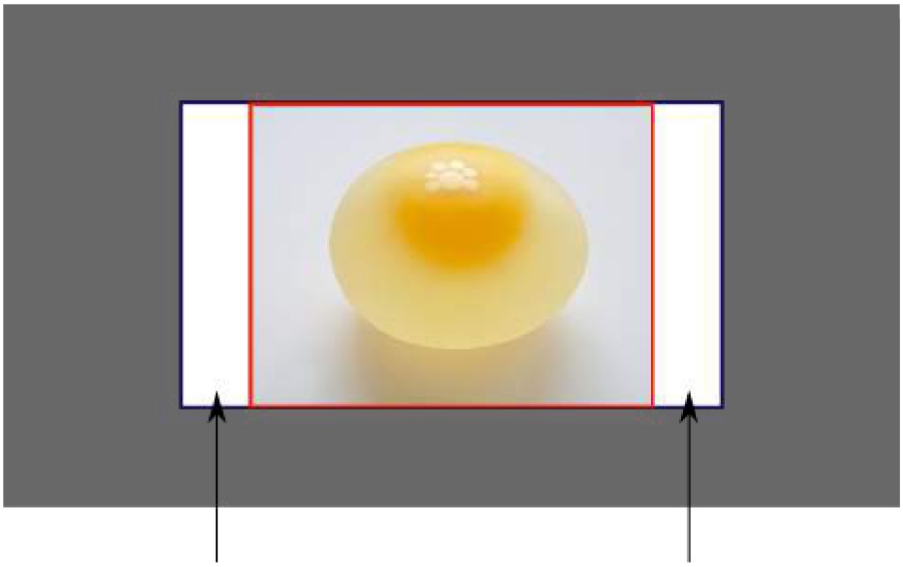
Sorry for the bad pun, but we're going to use a picture of the day by Biswarup Ganguly for demonstration. Lets say that this is your screen, and the gray area is outside of your visible screen. For demonstration, We're going to assume a 16 × 9 ratio.

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.

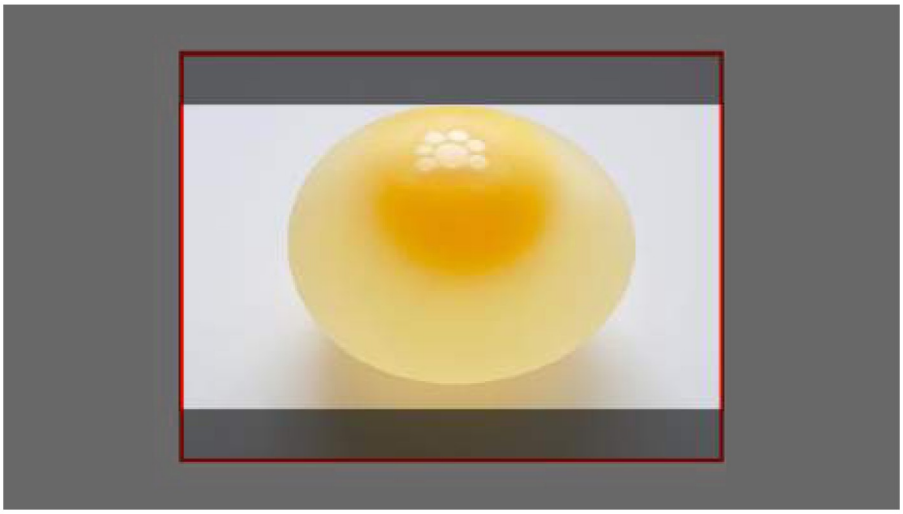
This makes sure that the background image is always completely contained in the background positioning area, however, there could be some empty space filled with your `background-color` in this case:



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>`.

Section 5.6: 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

	A percentage for the horizontal offset is relative to *(width of background positioning area - width of background image)*.
value% *value*%	A percentage for the vertical offset is relative to *(height of background positioning area - height of background image)* The size of the image is the size given by `background-size`.
*value*px *value*px	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.

Section 5.7: 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, consectetuer 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, consectetuer 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, consectetuer 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:

More:

https://www.w3.org/TR/css3-background/#the-background-origin

https://developer.mozilla.org/en-US/docs/Web/CSS/background-origin

Section 5.8: 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

Section 5.9: 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;
}
```

Section 5.10: Background Clip

Definition and Usage: The `background-clip` property specifies the painting area of the background.

Default value: `border-box`

Values

- `border-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, consectetuer 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, consectetuer 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, consectetuer 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>
```

Section 5.11: 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:

Section 5.12: 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;

Section 5.13: 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)";
```

Chapter 6: Centering

Section 6.1: Using Flexbox

HTML:

```
<div class="container">
  <img src="http://lorempixel.com/400/200" />
</div>
```

CSS:

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

View Result

HTML:

```
<img src="http://lorempixel.com/400/200" />
```

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.

Section 6.2: 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 caniuse.

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.

Section 6.3: Using margin: 0 auto;

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

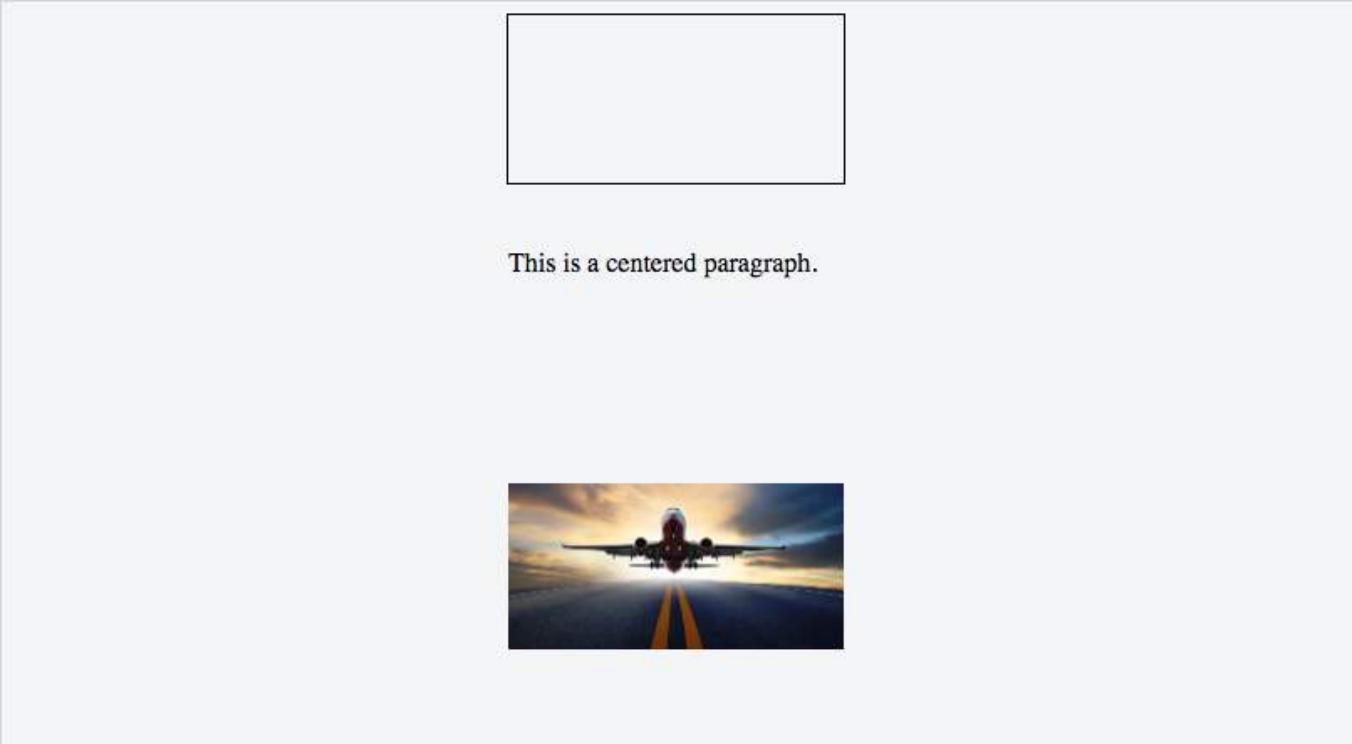
HTML

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

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

<div class="containerDiv">
    <img id="centeredImage"
src="https://i.kinja-img.com/gawker-media/image/upload/s--c7Q9b4Eh--/c_scale,fl_progressive,q_80,w_
800/qqyvc3bkpyl3mfhr8all.jpg" />
</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:



JSFiddle example: Centering objects with margin: 0 auto;

Section 6.4: 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.

Section 6.5: 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">
  <img class="center" src="http://lorempixel.com/400/200/" />
</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

Section 6.6: 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>
```

Section 6.7: 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.

Section 6.8: 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>
```

Section 6.9: 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">
      <img src="http://lorempixel.com/400/200/">
    </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 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.

Section 6.10: Ghost element technique (Michał Czernow'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;
}
```

```css
/* 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

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

Section 6.11: 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

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

CSS

```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**!

Section 6.12: Vertically align an image inside div

HTML

```
<div class="wrap">
    <img src="http://lorempixel.com/400/200/" />
</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;
}
```

Section 6.13: 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 */
}
```

Section 6.14: 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

Section 6.15: 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

```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;
}
```

Chapter 7: The Box Model

Parameter	Detail
`content-box`	Width and height of the element only includes content area.
`padding-box`	Width and height of the element includes content and padding.
`border-box`	Width and height of the element includes content, padding and border.
`initial`	Sets the box model to its default state.
`inherit`	Inherits the box model of the parent element.

Section 7.1: 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.

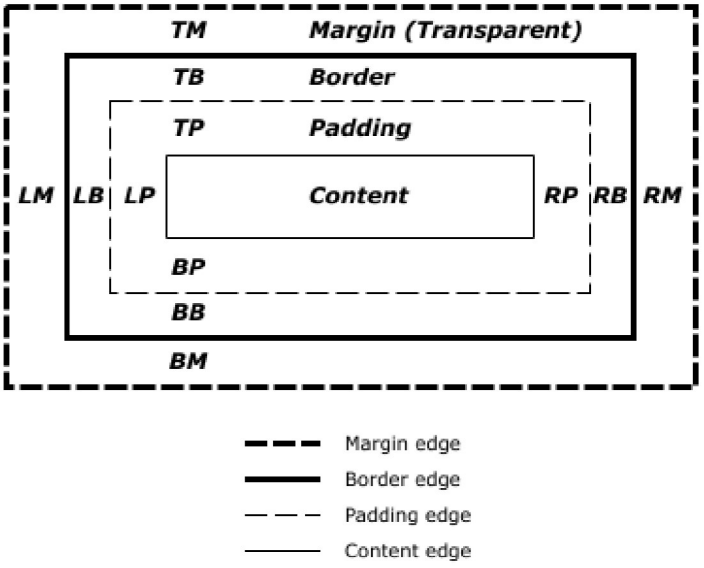


Diagram from CSS2.2 Working Draft

The perimeter of each of the four areas is called an *edge*. Each edge defines a *box.*

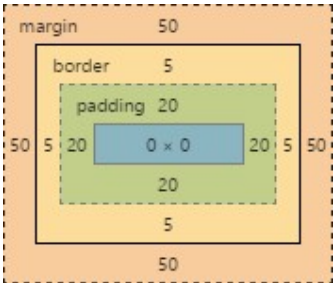
- The innermost rectangle is the **content box**. The width and height of this depends on the element's rendered content (text, images and any child elements it may have).
- Next is the **padding box**, as defined by the `padding` property. If there is no `padding` width defined, the padding edge is 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;
```

```
}
```

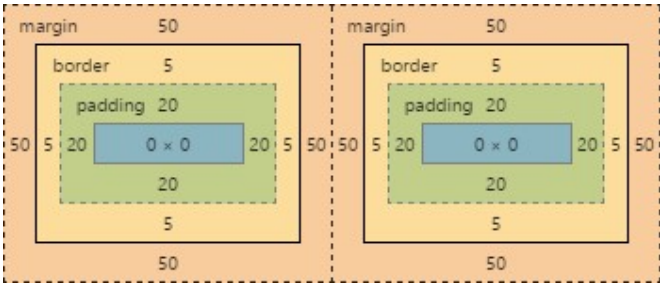
This CSS styles all `div` elements to have a top, right, bottom and left border of `5px` in width; a top, right, bottom and left margin of `50px`; and a top, right, bottom, and left padding of `20px`. Ignoring content, our generated box will look like this:



Screenshot of Google Chrome's Element Styles panel

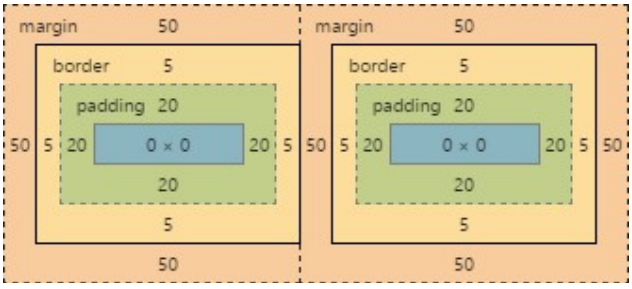
- As there is no content, the content region (the blue box in the middle) has no height or width (0px by 0px).
- The padding box by default is the same size as the content box, plus the 20px width on all four edges we're defining above with the `padding` property (40px by 40px).
- The border box is the 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:



Section 7.2: 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.

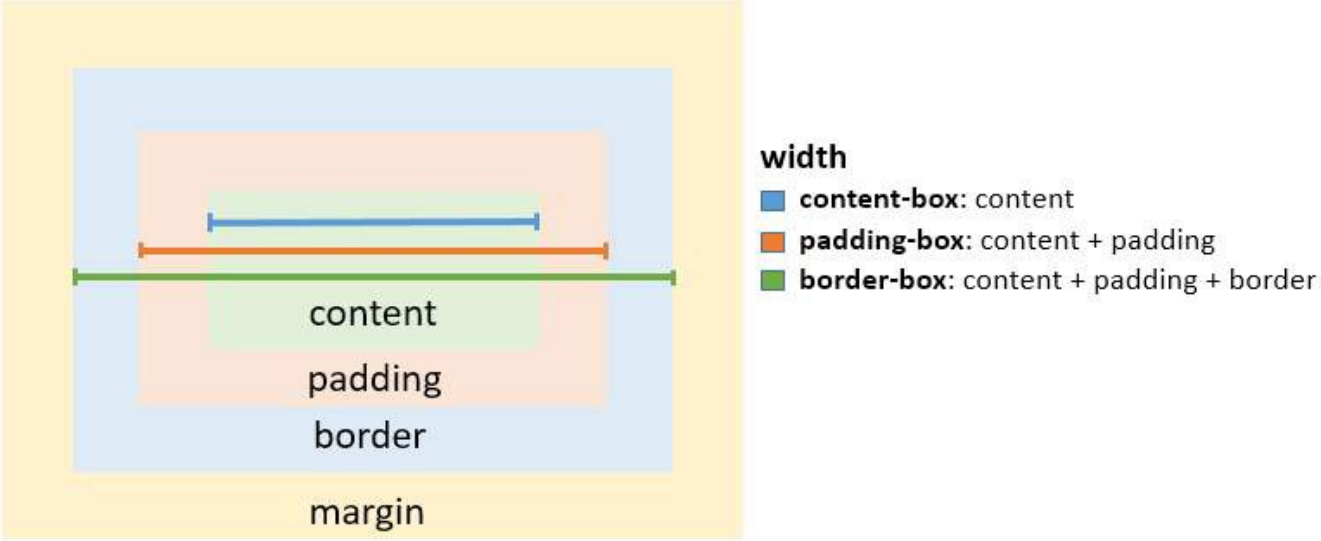
The following example demonstrates this potential issue with `content-box`:

```
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.

Chapter 8: Margins

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

Section 8.1: 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.

Section 8.2: 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
margin-left	The direction in which the margin should be applied.
30px	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;
}
```

Section 8.3: 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) */
}
```

Section 8.4: 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%

Section 8.5: 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 */
}
```

Section 8.6: 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>
```

Chapter 9: Padding

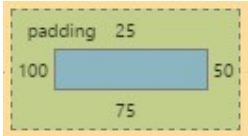
Section 9.1: Padding Shorthand

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:

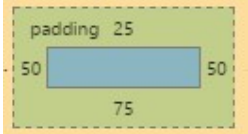
Four values:

```
<style>
    .myDiv {
        padding: 25px 50px 75px 100px; /* top right bottom left; */
    }
</style>
<div class="myDiv"></div>
```



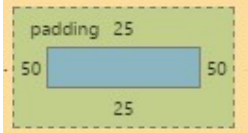
Three values:

```
<style>
    .myDiv {
        padding: 25px 50px 75px; /* top left/right bottom */
    }
</style>
<div class="myDiv"></div>
```



Two values:

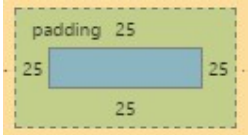
```
<style>
    .myDiv {
        padding: 25px 50px; /* top/bottom left/right */
    }
</style>
<div class="myDiv"></div>
```



One value:

```
<style>
    .myDiv {
```

```
        padding: 25px; /* top/right/bottom/left */
    }
</style>
<div class="myDiv"></div>
```



Section 9.2: 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:

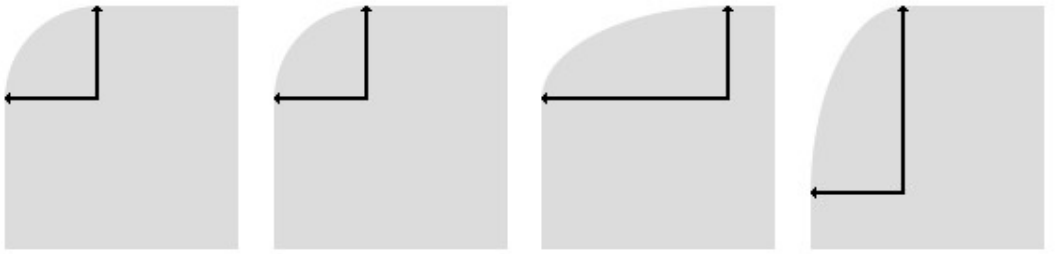
```
<style>
.myClass {
    padding-top: 5px;
}
</style>

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

Chapter 10: Border

Section 10.1: 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;
```

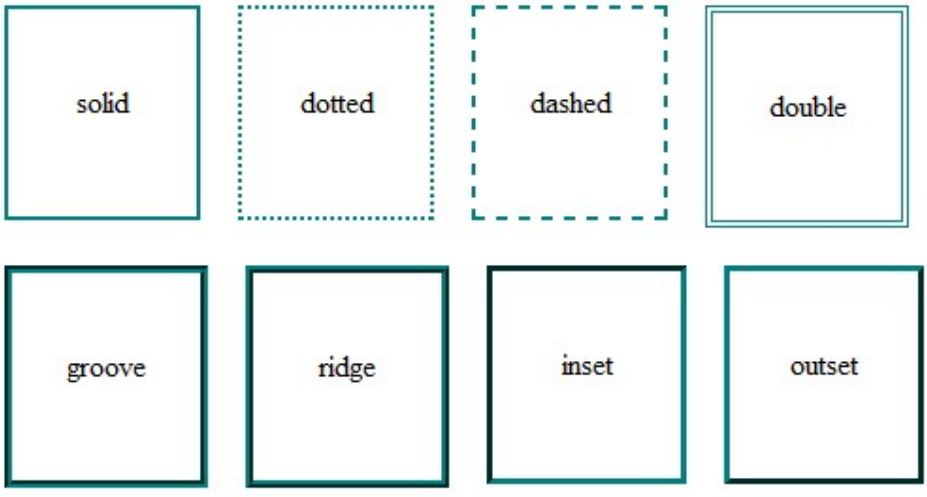
Section 10.2: border-style

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).

Section 10.3: Multiple Borders

Using outline:

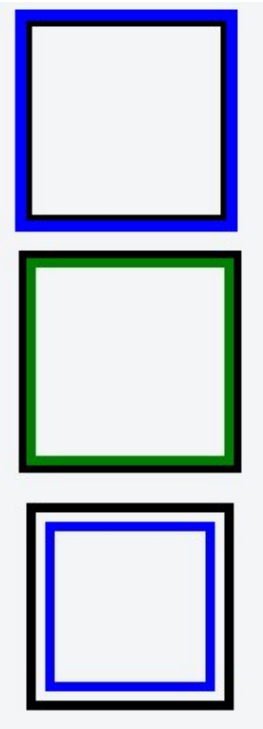
```
.div1{
  border: 3px solid black;
  outline: 6px solid blue;
  width: 100px;
  height: 100px;
  margin: 20px;
}
```

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/

Section 10.4: 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;
```

Section 10.5: border-collapse

The `border-collapse` property applies only to `table`s (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

Section 10.6: 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 wheras 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.

Section 10.7: Creating a multi-colored border using border-image

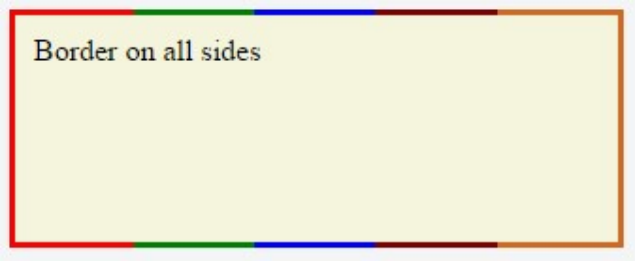
CSS

```
.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.

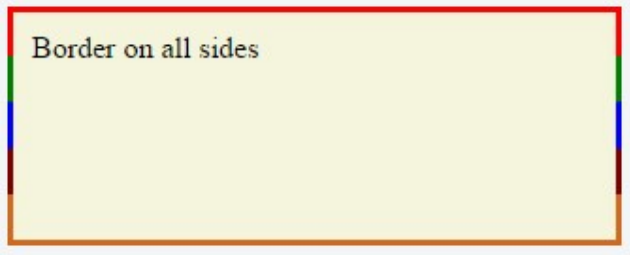


(*Note: Additional properties were added to the element for presentational purpose.*)

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.

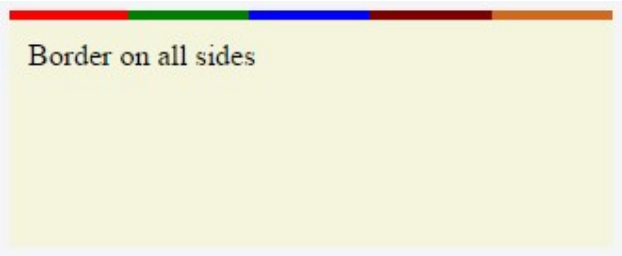
Which border(s) have a single color is dependant on the gradient definition. If the gradient is a `to right` gradient, the left border would be the start color of the gradient and right border would be the end color. If it was a `to bottom` gradient the top border would be the gradient's start color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

Section 10.8: 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;
}
```

Chapter 11: Outlines

Parameter	Details
dotted	dotted outline
dashed	dashed outline
solid	solid outline
double	double outline
groove	3D grooved outline, depends on the outline-color value
ridge	3D ridged outline, depends on the outline-color value
inset	3D inset outline, depends on the outline-color value
outset	3D outset outline, depends on the outline-color value
none	no outline
hidden	hidden outline

Section 11.1: 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;
```

Section 11.2: outline-style

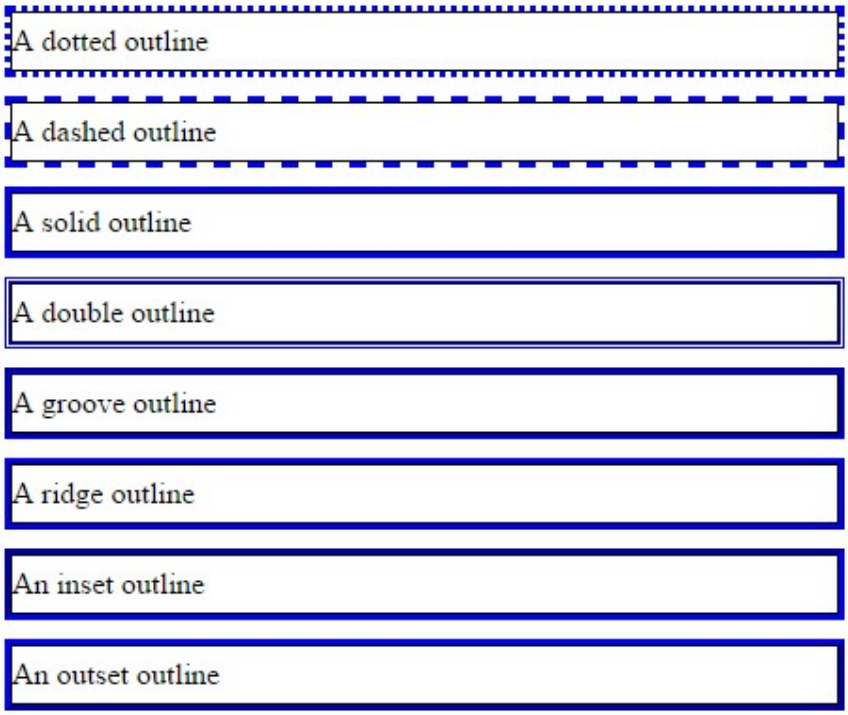
The `outline-style` property is used to set the style of the outline of an element.

```
p {
  border: 1px solid black;
  outline-color:blue;
  line-height:30px;
}
.p1{
  outline-style: dotted;
}
.p2{
  outline-style: dashed;
}
.p3{
  outline-style: solid;
}
```

```css
.p4{
  outline-style: double;
}
.p5{
  outline-style: groove;
}
.p6{
  outline-style: ridge;
}
.p7{
  outline-style: inset;
}
.p8{
  outline-style: outset;
}
```

HTML

```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>
```



Chapter 12: Overflow

`Overflow` Value	Details
`visible`	Shows all overflowing content outside the element
`scroll`	Hides the overflowing content and adds a scroll bar
`hidden`	Hides the overflowing content, both scroll bars disappear and the page becomes fixed
`auto`	Same as `scroll` if content overflows, but doesn't add scroll bar if content fits
`inherit`	Inherit's the parent element's value for this property

Section 12.1: overflow-wrap

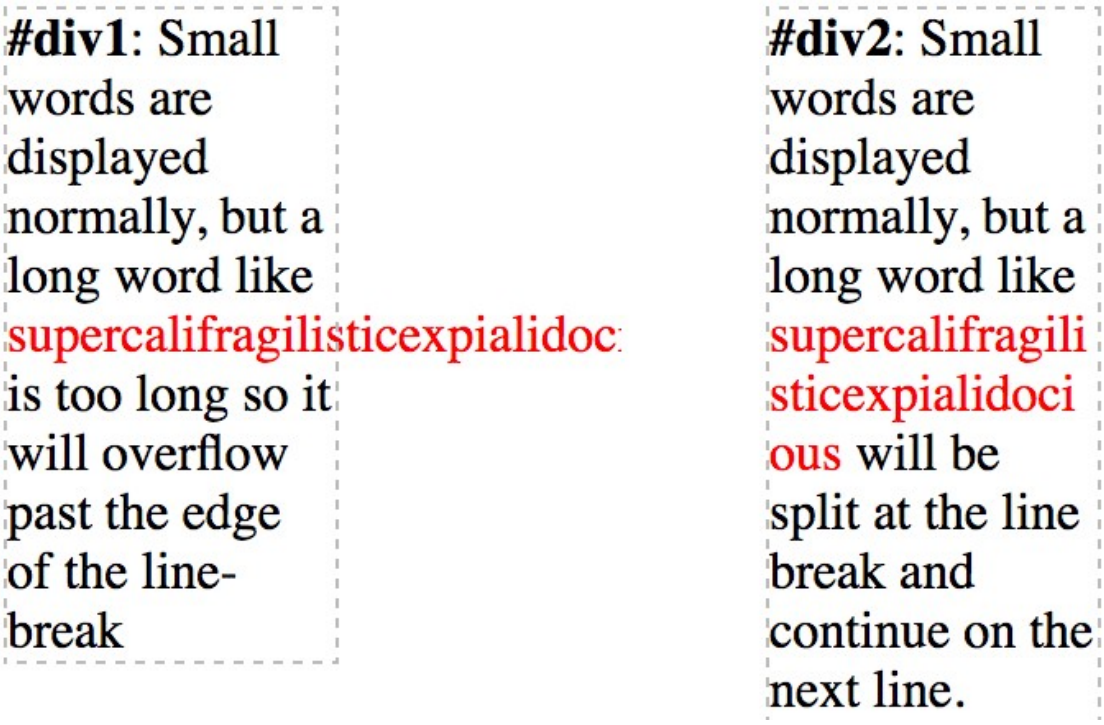
`overflow-wrap` tells a browser that it can break a line of text inside a targeted element onto multiple lines in an otherwise unbreakable place. Helpful in preventing an long string of text causing layout problems due to overflowing it's container.

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>
```



`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

Section 12.2: 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;
}
```

Section 12.3: overflow: scroll

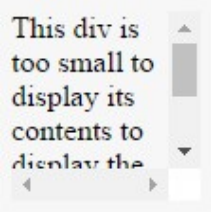
HTML

```
<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.

Section 12.4: 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.

Section 12.5: 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.

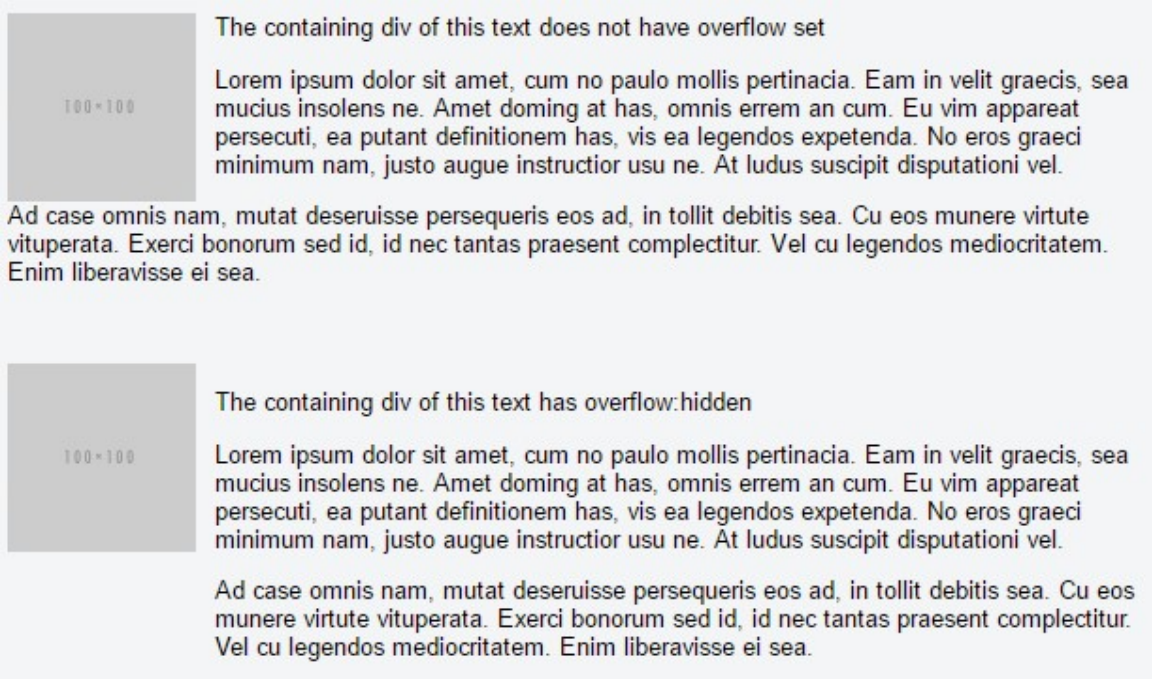
CSS

```
img {
    float:left;
    margin-right: 10px;
}
 div {
     overflow:hidden; /* creates block formatting context */
}
```

HTML

```
<img src="http://placehold.it/100x100">
<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.