## Starting with some HTML

Our starting point is an HTML document. You can copy the code from below if you want to work on your own computer. Save the code below as index.html in a folder on your machine.

<!doctype html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>Getting started with CSS</title>

</head>

<body>

<h1>I am a level one heading</h1>

<p>This is a paragraph of text. In the text is a <span>span element</span>

and also a <a href="http://example.com">link</a>.</p>

<p>This is the second paragraph. It contains an <em>emphasized</em> element.</p>

<ul>

<li>Item one</li>

<li>Item two</li>

<li>Item <em>three</em></li>

</ul>

</body>

</html>

**Note**: If you are reading this on a device or an environment where you can't easily create files, then don't worry — live code editors are provided below to allow you to write example code right here in the page.

## Adding CSS to our document

The very first thing we need to do is to tell the HTML document that we have some CSS rules we want it to use. There are three different ways to apply CSS to an HTML document that you'll commonly come across, however, for now, we will look at the most usual and useful way of doing so — linking CSS from the head of your document.

Create a file in the same folder as your HTML document and save it as styles.css. The .css extension shows that this is a CSS file.

To link styles.css to index.html add the following line somewhere inside the [<head>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/head) of the HTML document:

<link rel="stylesheet" href="styles.css">

This [<link>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/link) element tells the browser that we have a stylesheet, using the rel attribute, and the location of that stylesheet as the value of the href attribute. You can test that the CSS works by adding a rule to styles.css. Using your code editor add the following to your CSS file:

h1 {

color: red;

}

Save your HTML and CSS files and reload the page in a web browser. The level one heading at the top of the document should now be red. If that happens, congratulations — you have successfully applied some CSS to an HTML document. If that doesn't happen, carefully check that you've typed everything correctly.

You can continue to work in styles.css locally, or you can use our interactive editor below to continue with this tutorial. The interactive editor acts as if the CSS in the first panel is linked to the HTML document, just as we have with our document above.

## Styling HTML elements

By making our heading red we have already demonstrated that we can target and style an HTML element. We do this by targeting an element selector — this is a selector that directly matches an HTML element name. To target all paragraphs in the document you would use the selector p. To turn all paragraphs green you would use:

p {

color: green;

}

You can target multiple selectors at once, by separating the selectors with a comma. If I want all paragraphs and all list items to be green my rule looks like this:

p, li {

color: green;

}

Try this out in the interactive editor below (edit the code boxes), or in your local CSS document.

## Changing the default behavior of elements

When we look at a well-marked up HTML document, even something as simple as our example, we can see how the browser is making the HTML readable by adding some default styling. Headings are large and bold and our list has bullets. This happens because browsers have internal stylesheets containing default styles, which they apply to all pages by default; without them all of the text would run together in a clump and we would have to style everything from scratch. All modern browsers display HTML content by default in pretty much the same way.

However, you will often want something other than the choice the browser has made. This can be done by simply choosing the HTML element that you want to change, and using a CSS rule to change the way it looks.  A good example is our <ul>, an unordered list. It has list bullets, and if I decide I don't want those bullets I can remove them like so:

li {

list-style-type: none;

}

Try adding this to your CSS now.

The list-style-type property is a good property to look at on MDN to see which values are supported. Take a look at the page for [list-style-type](https://developer.mozilla.org/en-US/docs/Web/CSS/list-style-type) and you will find an interactive example at the top of the page to try some different values in, then all allowable values are detailed further down the page.

Looking at that page you will discover that in addition to removing the list bullets you can change them — try changing them to square bullets by using a value of square.

So far we have styled elements based on their HTML element names. This works as long as you want all of the elements of that type in your document to look the same. Most of the time that isn't the case and so you will need to find a way to select a subset of the elements without changing the others. The most common way to do this is to add a class to your HTML element and target that class.

In your HTML document, add a [class attribute](https://developer.mozilla.org/en-US/docs/Web/HTML/Global_attributes/class) to the second list item. Your list will now look like this:

<ul>

<li>Item one</li>

<li **class="special"**>Item two</li>

<li>Item <em>three</em></li>

</ul>

In your CSS you can target the class of special by creating a selector that starts with a full stop character. Add the following to your CSS file:

.special {

color: orange;

font-weight: bold;

}

Save and refresh to see what the result is.

You can apply the class of special to any element on your page that you want to have the same look as this list item. For example, you might want the <span> in the paragraph to also be orange and bold. Try adding a class of special to it, then reload your page and see what happens.

Sometimes you will see rules with a selector that lists the HTML element selector along with the class:

li.special {

color: orange;

font-weight: bold;

}

This syntax means "target any li element that has a class of special". If you were to do this then you would no longer be able to apply the class to a <span> or another element by simply adding the class to it; you would have to add that element to the list of selectors:

li.special,

span.special {

color: orange;

font-weight: bold;

}

As you can imagine, some classes might be applied to many elements and you don't want to have to keep editing your CSS every time something new needs to take on that style. Therefore it is sometimes best to bypass the element and simply refer to the class, unless you know that you want to create some special rules for one element alone, and perhaps want to make sure they are not applied to other things.

## Styling things based on their location in a document

There are times when you will want something to look different based on where it is in the document. There are a number of selectors that can help you here, but for now we will look at just a couple. In our document are two <em> elements — one inside a paragraph and the other inside a list item. To select only an <em> that is nested inside an <li> element I can use a selector called the **descendant combinator**, which simply takes the form of a space between two other selectors.

Add the following rule to your stylesheet.

li em {

color: rebeccapurple;

}

This selector will select any <em> element that is inside (a descendant of) an <li>. So in your example document, you should find that the <em> in the third list item is now purple, but the one inside the paragraph is unchanged.

Something else you might like to try is styling a paragraph when it comes directly after a heading at the same hierarchy level in the HTML. To do so place a +  (an **adjacent sibling combinator**) between the selectors.

Try adding this rule to your stylesheet as well:

h1 + p {

font-size: 200%;

}

The live example below includes the two rules above. Try adding a rule to make a span red, if it is inside a paragraph. You will know if you have it right as the span in the first paragraph will be red, but the one in the first list item will not change color.

**Note**: As you can see, CSS gives us several ways to target elements, and we've only scratched the surface so far! We will be taking a proper look at all of these selectors and many more in our [Selectors](https://developer.mozilla.org/en-US/docs/Learn/CSS/Building_blocks/Selectors) articles later on in the course.

## Styling things based on state

The final type of styling we shall take a look at in this tutorial is the ability to style things based on their state. A straightforward example of this is when styling links. When we style a link we need to target the [<a>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/a) (anchor) element. This has different states depending on whether it is unvisited, visited, being hovered over, focused via the keyboard, or in the process of being clicked (activated). You can use CSS to target these different states — the CSS below styles unvisited links pink and visited links green.

a:link {

color: pink;

}

a:visited {

color: green;

}

You can change the way the link looks when the user hovers over it, for example removing the underline, which is achieved by in the next rule:

a:hover {

text-decoration: none;

}

In the live example below, you can play with different values for the various states of a link. I have added the rules above to it, and now realise that the pink color is quite light and hard to read — why not change that to a better color? Can you make the links bold?

We have removed the underline on our link on hover. You could remove the underline from all states of a link. It is worth remembering however that in a real site, you want to ensure that visitors know that a link is a link. Leaving the underline in place, can be an important clue for people to realize that some text inside a paragraph can be clicked on — this is the behavior they are used to. As with everything in CSS, there is the potential to make the document less accessible with your changes — we will aim to highlight potential pitfalls in appropriate places.

**Note**: you will often see mention of [accessibility](https://developer.mozilla.org/en-US/docs/Learn/Accessibility) in these lessons and across MDN. When we talk about accessibility we are referring to the requirement for our webpages to be understandable and usable by everyone.

Your visitor may well be on a computer with a mouse or trackpad, or a phone with a touchscreen. Or they might be using a screenreader, which reads out the content of the document, or they may need to use much larger text, or be navigating the site using the keyboard only.

A plain HTML document is generally accessible to everyone — as you start to style that document it is important that you don't make it less accessible.

## Combining selectors and combinators

It is worth noting that you can combine multiple selectors and combinators together. For example:

/\* selects any <span> that is inside a <p>, which is inside an <article> \*/

article p span { ... }

/\* selects any <p> that comes directly after a <ul>, which comes directly after an <h1> \*/

h1 + ul + p { ... }

You can combine multiple types together, too. Try adding the following into your code:

body h1 + p .special {

color: yellow;

background-color: black;

padding: 5px;

}

This will style any element with a class of special, which is inside a <p>, which comes just after an <h1>, which is inside a <body>. Phew!

In the original HTML we provided, the only element styled is <span class="special">.

Don't worry if this seems complicated at the moment — you'll soon start to get the hang of it as you write more CSS.

You can apply **multiple backgrounds** to elements. These are layered atop one another with the first background you provide on top and the last background listed in the back. Only the last background can include a background color.

Specifying multiple backgrounds is easy:

.myclass {

background: background1, background 2, ..., backgroundN;

}

You can do this with both the shorthand [background](https://developer.mozilla.org/en-US/docs/Web/CSS/background) property and the individual properties thereof except for [background-color](https://developer.mozilla.org/en-US/docs/Web/CSS/background-color). That is, the following background properties can be specified as a list, one per background: [background](https://developer.mozilla.org/en-US/docs/Web/CSS/background), [background-attachment](https://developer.mozilla.org/en-US/docs/Web/CSS/background-attachment), [background-clip](https://developer.mozilla.org/en-US/docs/Web/CSS/background-clip), [background-image](https://developer.mozilla.org/en-US/docs/Web/CSS/background-image), [background-origin](https://developer.mozilla.org/en-US/docs/Web/CSS/background-origin), [background-position](https://developer.mozilla.org/en-US/docs/Web/CSS/background-position), [background-repeat](https://developer.mozilla.org/en-US/docs/Web/CSS/background-repeat), [background-size](https://developer.mozilla.org/en-US/docs/Web/CSS/background-size).

**Example**

In this example, three backgrounds are stacked: the Firefox logo, an image of bubbles, and a [linear gradient](https://developer.mozilla.org/en-US/docs/Web/CSS/linear-gradient):

HTML

<div class="multi-bg-example"></div>

CSS

.multi-bg-example {

width: 100%;

height: 400px;

background-image: url(https://mdn.mozillademos.org/files/11305/firefox.png),

url(https://mdn.mozillademos.org/files/11307/bubbles.png),

linear-gradient(to right, rgba(30, 75, 115, 1), rgba(255, 255, 255, 0));

background-repeat: no-repeat,

no-repeat,

no-repeat;

background-position: bottom right,

left,

right;

}

Result

(If image does not appear in CodePen, click the 'Tidy' button in the CSS section)

As you can see here, the Firefox logo (listed first within [background-image](https://developer.mozilla.org/en-US/docs/Web/CSS/background-image)) is on top, directly above the bubbles graphic, followed by the gradient (listed last) sitting underneath all previous 'images'. Each subsequent sub-property ([background-repeat](https://developer.mozilla.org/en-US/docs/Web/CSS/background-repeat) and [background-position](https://developer.mozilla.org/en-US/docs/Web/CSS/background-position)) applies to the corresponding backgrounds. So the first listed value for [background-repeat](https://developer.mozilla.org/en-US/docs/Web/CSS/background-repeat) applies to the first (frontmost) background, and so forth.

## Applying CSS to your HTML

The first thing we will look at are the three methods of applying CSS to a document.

### External stylesheet

In the [Getting started with CSS](https://developer.mozilla.org/en-US/docs/Learn/CSS/First_steps/Getting_started) we linked an external stylesheet to our page. This is the most common and useful method of attaching CSS to a document as you can link the CSS to multiple pages, allowing you to style them all with the same stylesheet. In most cases, the different pages of a site will all look pretty much the same, therefore you can use the same set of rules for the basic look and feel.

An external stylesheet is when you have your CSS written in a separate file with a .css extension, and you reference it from an HTML <link> element:

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>My CSS experiment</title>

<link rel="stylesheet" href="styles.css">

</head>

<body>

<h1>Hello World!</h1>

<p>This is my first CSS example</p>

</body>

</html>

The CSS file might look something like this:

h1 {

color: blue;

background-color: yellow;

border: 1px solid black;

}

p {

color: red;

}

The href attribute of the [<link>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/link) element needs to reference a file on your filesystem.

In the example above, the CSS file is in the same folder as the HTML document, but you could place it somewhere else and adjust the specified path to suit, for example:

<!-- Inside a subdirectory called styles inside the current directory -->

<link rel="stylesheet" href="styles/style.css">

<!-- Inside a subdirectory called general, which is in a subdirectory called styles, inside the current directory -->

<link rel="stylesheet" href="styles/general/style.css">

<!-- Go up one directory level, then inside a subdirectory called styles -->

<link rel="stylesheet" href="../styles/style.css">

### Internal stylesheet

An internal stylesheet is where you don't have an external CSS file, but instead place your CSS inside a [<style>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/style) element contained inside the HTML [<head>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/head).

So the HTML would look like this:

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>My CSS experiment</title>

<style>

h1 {

color: blue;

background-color: yellow;

border: 1px solid black;

}

p {

color: red;

}

</style>

</head>

<body>

<h1>Hello World!</h1>

<p>This is my first CSS example</p>

</body>

</html>

This can be useful in some circumstances (maybe you're working with a content management system where you can't modify the CSS files directly), but it isn't quite as efficient as external stylesheets — in a website, the CSS would need to be repeated across every page, and updated in multiple places if changes were required.

### Inline styles

Inline styles are CSS declarations that affect one element only, contained within a style attribute:

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>My CSS experiment</title>

</head>

<body>

<h1 style="color: blue;background-color: yellow;border: 1px solid black;">Hello World!</h1>

<p style="color:red;">This is my first CSS example</p>

</body>

</html>

**Please don't do this, unless you really have to!** It is really bad for maintenance (you might have to update the same information multiple times per document), and it also mixes your presentational CSS information with your HTML structural information, making the code harder to read and understand. Keeping different types of code separated makes for a much easier job for all who work on the code.

There are a few places where inline styles are more common, or even advisable. You might have to resort to using them if your working environment is really restrictive (perhaps your CMS only allows you to edit the HTML body). You will also see them used a lot in HTML email in order to get compatibility with as many email clients as possible.

## Playing with the CSS in this article

There is a lot of CSS to play with in this article. To do so, we'd recommend creating a new directory/folder on your computer, and inside it creating a copy of the following two files:

**index.html:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>My CSS experiments</title>

<link rel="stylesheet" href="styles.css">

</head>

<body>

<p>Create your test HTML here</p>

</body>

</html>

**styles.css:**

/\* Create your test CSS here \*/

p {

color: red;

}

Then, when you come across some CSS you want to experiment with, replace the HTML <body> contents with some HTML to style, and start adding CSS to style it inside your CSS file.

If you are not using a system where you can easily create files, you can instead use the interactive editor below to experiment.

Read on, and have fun!

## Selectors

You can't talk about CSS without meeting selectors, and we have already discovered several different types in the [Getting started with CSS](https://developer.mozilla.org/en-US/docs/Learn/CSS/First_steps/Getting_started) tutorial. A selector is how we target something in our HTML document in order to apply styles to it. If your styles are not applying then it is likely that your selector does not match the thing you think it should match.

Each CSS rule starts with a selector or a list of selectors in order to tell the browser which element or elements the rules should apply to. All of the following are examples of valid selectors, or lists of selectors.

h1

a:link

.manythings

#onething

\*

.box p

.box p:first-child

h1, h2, .intro

**Try creating some CSS rules that use the above selectors, and some HTML to be styled by them. If you don't know what some of the above syntax means, try searching for it on MDN!**

**Note**: You will learn a lot more about selectors in our [CSS selectors](https://developer.mozilla.org/en-US/docs/Learn/CSS/Building_blocks/Selectors) tutorials, in the next module.

### Specificity

There will often be scenarios where two selectors could select the same HTML element. Consider the stylesheet below where I have a rule with a p selector that will set paragraphs to blue, and also a class that will set selected elements red.

.special {

color: red;

}

p {

color: blue;

}

Let's say that in our HTML document we have a paragraph with a class of special. Both rules could apply, so which one wins? What color do you think our paragraph will become?

<p class="special">What color am I?</p>

The CSS language has rules to control which rule will win in the event of a collision — these are called **cascade** and **specificity**. In the below code block we have defined two rules for the p selector, but the paragraph ends up being colored blue. This is because the declaration that sets it to blue appears later in the stylesheet, and later styles override earlier ones. This is the cascade in action.

p {

color: red;

}

p {

color: blue;

}

However, in the case of our earlier block with the class selector and the element selector, the class will win, making the paragraph red — even thought it appears earlier in the stylesheet. A class is described as being more specific, or having more specificity than the element selector, so it wins.

**Try the above experiment for yourself — add the HTML to your experiment, then add the two p { ... } rules to your stylesheet. Next, change the first p selector to .special to see how it changes the styling.**

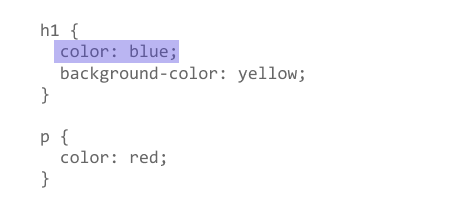
The rules of specificity and the cascade can seem a little complicated at first and are easier to understand once you have built up further CSS knowledge. In our [Cascade and inheritance](https://developer.mozilla.org/en-US/docs/Learn/CSS/Building_blocks/Cascade_and_inheritance) article, which you'll get to in the next module, I'll explain this in detail, including how to calculate specificity. For now, remember that this exists, and that sometimes CSS might not apply like you expect it to because something else in your stylesheet has a higher specificity. Identifying that more than one rule could apply to an element is the first step in fixing such issues.

## Properties and values

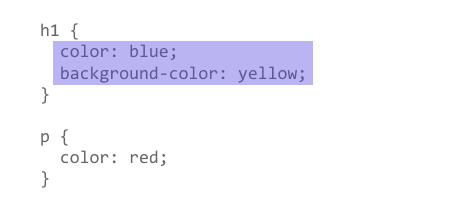
At its most basic level, CSS consists of two building blocks:

* **Properties**: Human-readable identifiers that indicate which stylistic features (e.g. [font-size](https://developer.mozilla.org/en-US/docs/Web/CSS/font-size), [width](https://developer.mozilla.org/en-US/docs/Web/CSS/width), [background-color](https://developer.mozilla.org/en-US/docs/Web/CSS/background-color)) you want to change.
* **Values**: Each specified property is given a value, which indicates how you want to change those stylistic features (e.g. what you want to change the font, width or background color to.)

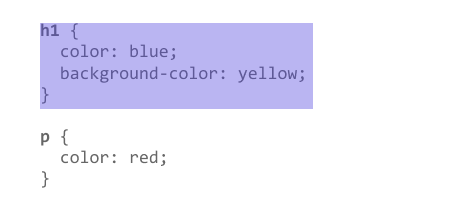
The below image highlights a single property and value. The property name is color, and the value blue.



A property paired with a value is called a CSS declaration. CSS declarations are put within CSS Declaration Blocks. This next image shows our CSS with the declaration block highlighted.



Finally, CSS declaration blocks are paired with selectors to produce CSS Rulesets (or CSS Rules). Our image contains two rules, one for the h1 selector and one for the p selector. The rule for h1 is highlighted.



Setting CSS properties to specific values is the core function of the CSS language. The CSS engine calculates which declarations apply to every single element of a page in order to appropriately lay it out and style it. What is important to remember is that both properties and values are case-sensitive in CSS. The property and value in each pair is separated by a colon (:).

**Try looking up different values of the following properties, and writing CSS rules that apply them to different HTML elements:**

* [**font-size**](https://developer.mozilla.org/en-US/docs/Web/CSS/font-size)
* [**width**](https://developer.mozilla.org/en-US/docs/Web/CSS/width)
* [**background-color**](https://developer.mozilla.org/en-US/docs/Web/CSS/background-color)
* [**color**](https://developer.mozilla.org/en-US/docs/Web/CSS/color)
* [**border**](https://developer.mozilla.org/en-US/docs/Web/CSS/border)

**Important**: If a property is unknown or if a value is not valid for a given property, the declaration is deemed invalid and is completely ignored by the browser's CSS engine.

**Important**: In CSS (and other web standards), US spelling has been agreed on as the standard to stick to where language uncertainty arises. For example, color should always be spelled color. colour won't work.

### Functions

While most values are relatively simple keywords or numeric values, there are some possible values which take the form of a function. An example would be the calc() function. This function allows you to do simple math from within your CSS, for example:

<div class="outer"><div class="box">The inner box is 90% - 30px.</div></div>

.outer {

border: 5px solid black;

}

.box {

padding: 10px;

width: calc(90% - 30px);

background-color: rebeccapurple;

color: white;

}

This renders like so:

Open in CodePenOpen in JSFiddle

A function consists of the function name, and then some brackets into which the allowed values for that function are placed. In the case of the calc() example above I am asking for the width of this box to be 90% of the containing block width, minus 30 pixels. This isn't something I can calculate ahead of time and just enter the value into the CSS, as I don't know what 90% will be. As with all values, the relevant page on MDN will have usage examples so you can see how the function works.

Another example would be the various values for [transform](https://developer.mozilla.org/en-US/docs/Web/CSS/transform), such as rotate().

<div class="box"></div>

.box {

margin: 30px;

width: 100px;

height: 100px;

background-color: rebeccapurple;

transform: rotate(0.8turn)

}

The output from the above code looks like this:

Open in CodePenOpen in JSFiddle

**Try looking up different values of the following properties, and writing CSS rules that apply them to different HTML elements:**

* [**transform**](https://developer.mozilla.org/en-US/docs/Web/CSS/transform)
* [**background-image**](https://developer.mozilla.org/en-US/docs/Web/CSS/background-image)**, in particular gradient values**
* [**color**](https://developer.mozilla.org/en-US/docs/Web/CSS/color)**, in particular rgb/rgba/hsl/hsla values**

## @rules

As yet, we have not encountered [@rules](https://developer.mozilla.org/en-US/docs/Web/CSS/At-rule) (pronounced "at-rules"). These are special rules giving CSS some instruction on how to behave. Some @rules are simple with the rule name and a value. For example, to import an additional stylesheet into your main CSS stylesheet you can use @import:

@import 'styles2.css';

One of the most common @rules you will come across is @media, which allows you to use [media queries](https://developer.mozilla.org/en-US/docs/Web/CSS/Media_Queries) to apply CSS only when certain conditions are true (e.g. when the screen resolution is above a certain amount, or the screen is wider than a certain width).

In the below CSS, we have a stylesheet that gives the <body> element a pink background color. However, we then use @media to create a section of our stylesheet that will only be applied in browsers with a viewport wider than 30em. If the browser is wider than 30em then the background color will be blue.

body {

background-color: pink;

}

@media (min-width: 30em) {

body {

background-color: blue;

}

}

You will encounter other @rules throughout these tutorials.

**See if you can add a media query to your CSS that changes styles based on the viewport width. Change the width of your browser window to see the result.**

## Shorthands

Some properties like [font](https://developer.mozilla.org/en-US/docs/Web/CSS/font), [background](https://developer.mozilla.org/en-US/docs/Web/CSS/background), [padding](https://developer.mozilla.org/en-US/docs/Web/CSS/padding), [border](https://developer.mozilla.org/en-US/docs/Web/CSS/border), and [margin](https://developer.mozilla.org/en-US/docs/Web/CSS/margin) are called **shorthand properties** — this is because they allow you to set several property values in a single line, saving time and making your code neater in the process.

For example, this line:

/\* In 4-value shorthands like padding and margin, the values are applied

in the order top, right, bottom, left (clockwise from the top). There are also other

shorthand types, for example 2-value shorthands, which set padding/margin

for top/bottom, then left/right \*/

padding: 10px 15px 15px 5px;

Does the same thing as all these together:

padding-top: 10px;

padding-right: 15px;

padding-bottom: 15px;

padding-left: 5px;

Whereas this line:

background: red url(bg-graphic.png) 10px 10px repeat-x fixed;

Does the same thing as all these together:

background-color: red;

background-image: url(bg-graphic.png);

background-position: 10px 10px;

background-repeat: repeat-x;

background-attachment: fixed;

We won't attempt to teach these exhaustively now — you'll come across many examples later on in the course, and you are advised to look up the shorthand property names in our [CSS reference](https://developer.mozilla.org/en-US/docs/Web/CSS/Reference) to find out more.

**Try adding the above declarations to your CSS to see how it affects the styling of your HTML. Try experimenting with some different values.**

**Warning**: While shorthands often allow you to miss out values, they will then reset any values that you do not include to their initial values. This ensures that a sensible set of values are used. However, this might be confusing if you were expecting the shorthand to only change the values you passed in.

## Comments

As with HTML, you are encouraged to make comments in your CSS, to help you understand how your code works when coming back to it after several months, and to help others coming to the code to work on it understand it.

Comments in CSS begin with /\* and end with \*/. In the below code block I have used comments to mark the start of different distinct code sections. This is useful to help you navigate your codebase as it gets larger — you can search for the comments in your code editor.

/\* Handle basic element styling \*/

/\* -------------------------------------------------------------------------------------------- \*/

body {

font: 1em/150% Helvetica, Arial, sans-serif;

padding: 1em;

margin: 0 auto;

max-width: 33em;

}

@media (min-width: 70em) {

/\* Let's special case the global font size. On large screen or window,

we increase the font size for better readability \*/

body {

font-size: 130%;

}

}

h1 {font-size: 1.5em;}

/\* Handle specific elements nested in the DOM \*/

/\* -------------------------------------------------------------------------------------------- \*/

div p, #id:first-line {

background-color: red;

border-radius: 3px;

}

div p {

margin: 0;

padding: 1em;

}

div p + p {

padding-top: 0;

}

Comments are also useful for temporarily commenting out certain parts of the code for testing purposes, for example if you are trying to find which part of your code is causing an error. In the next example I have commented out the rules for the .special selector.

/\*.special {

color: red;

}\*/

p {

color: blue;

}

**Add some comments to your CSS, to get used to using them.**

## Whitespace

White space means actual spaces, tabs and new lines. In the same manner as HTML, the browser tends to ignore much of the whitespace inside your CSS; a lot of the whitespace is just there to aid readability.

In our first example below we have each declaration (and rule start/end) on its own line — this is arguably a good way to write CSS, as it makes it easy to maintain and understand:

body {

font: 1em/150% Helvetica, Arial, sans-serif;

padding: 1em;

margin: 0 auto;

max-width: 33em;

}

@media (min-width: 70em) {

body {

font-size: 130%;

}

}

h1 {

font-size: 1.5em;

}

div p,

#id:first-line {

background-color: red;

border-radius: 3px;

}

div p {

margin: 0;

padding: 1em;

}

div p + p {

padding-top: 0;

}

You could write exactly the same CSS like so, with most of the whitespace removed — this is functionally identical to the first example, but I'm sure you'll agree that it is somewhat harder to read:

body {font: 1em/150% Helvetica, Arial, sans-serif; padding: 1em; margin: 0 auto; max-width: 33em;}

@media (min-width: 70em) { body {font-size: 130%;} }

h1 {font-size: 1.5em;}

div p, #id:first-line {background-color: red; border-radius: 3px;}

div p {margin: 0; padding: 1em;}

div p + p {padding-top: 0;}

The code layout you choose is usually a personal preference, although when you start to work in teams, you may find that the existing team has its own styleguide that specifies an agreed convention to follow.

The whitespace between the properties and their values you need to be careful of in CSS.

For example, the following declarations are valid CSS:

margin: 0 auto;

padding-left: 10px;

But the following are invalid:

margin: 0auto;

padding- left: 10px;

0auto is not recognised as a valid value for the margin property (0 and auto are two separate values,) and the browser does not recognise padding- as a valid property. So you should always make sure to separate distinct values from one another by at least a space, but keep property names and property values together as single unbroken strings.

**Try playing with whitespace inside your CSS, to see what breaks things and what doesn't.**

**How does CSS actually work?**

When a browser displays a document, it must combine the document's content with its style information. It processes the document in a number of stages, which we've listed below. Bear in mind that this is a very simplified version of what happens when a browser loads a webpage, and that different browsers will handle the process in different ways. But this is roughly what happens.

1. The browser loads the HTML (e.g. receives it from the network).
2. It converts the [HTML](https://developer.mozilla.org/en-US/docs/Glossary/HTML) into a [DOM](https://developer.mozilla.org/en-US/docs/Glossary/DOM) (*Document Object Model*). The DOM represents the document in the computer's memory. The DOM is explained in a bit more detail in the next section.
3. The browser then fetches most of the resources that are linked to by the HTML document, such as embedded images and videos ... and linked CSS! JavaScript is handled a bit later on in the process, and we won't talk about it here to keep things simpler.
4. The browser parses the fetched CSS, and sorts the different rules by their selector types into different "buckets", e.g. element, class, ID, and so on. Based on the selectors it finds, it works out which rules should be applied to which nodes in the DOM, and attaches style to them as required (this intermediate step is called a render tree).
5. The render tree is laid out in the structure it should appear in after the rules have been applied to it.
6. The visual display of the page is shown on the screen (this stage is called painting).

The following diagram also offers a simple view of the process.

**About the DOM**

A DOM has a tree-like structure. Each element, attribute, and piece of text in the markup language becomes a [DOM node](https://developer.mozilla.org/en-US/docs/Glossary/Node/DOM) in the tree structure. The nodes are defined by their relationship to other DOM nodes. Some elements are parents of child nodes, and child nodes have siblings.

Understanding the DOM helps you design, debug and maintain your CSS because the DOM is where your CSS and the document's content meet up. When you start working with browser DevTools you will be navigating the DOM as you select items in order to see which rules apply.

**A real DOM representation**

Rather than a long, boring explanation, let's look at an example to see how a real HTML snippet is converted into a DOM.

Take the following HTML code:

<p>

Let's use:

<span>Cascading</span>

<span>Style</span>

<span>Sheets</span>

</p>

In the DOM, the node corresponding to our <p> element is a parent. Its children are a text node and the three nodes corresponding to our <span> elements. The SPAN nodes are also parents, with text nodes as their children:

P

├─ "Let's use:"

├─ SPAN

| └─ "Cascading"

├─ SPAN

| └─ "Style"

└─ SPAN

└─ "Sheets"

This is how a browser interprets the previous HTML snippet —it renders the above DOM tree and then outputs it in the browser like so:

**Applying CSS to the DOM**

Let's say we added some CSS to our document, to style it. Again, the HTML is as follows:

<p>

Let's use:

<span>Cascading</span>

<span>Style</span>

<span>Sheets</span>

</p>

Let's suppose we apply the following CSS to it:

span {

border: 1px solid black;

background-color: lime;

}

The browser will parse the HTML and create a DOM from it, then parse the CSS. Since the only rule available in the CSS has a span selector, the browser will be able to sort the CSS very quickly! It will apply that rule to each one of the three <span>s, then paint the final visual representation to the screen.

The updated output is as follows:

In our [Debugging CSS](https://developer.mozilla.org/en-US/docs/Learn/CSS/Building_blocks/Debugging_CSS) article in the next module we will be using browser DevTools to debug CSS problems, and will learn more about how the browser interprets CSS.

**What happens if a browser encounters CSS it doesn't understand?**

[In an earlier lesson](https://developer.mozilla.org/en-US/docs/Learn/CSS/First_steps/What_is_CSS#Browser_support) I mentioned that browsers do not all implement new CSS at the same time. In addition, many people are not using the latest version of a browser. Given that CSS is being developed all the time, and is therefore ahead of what browsers can recognise, you might wonder what happens if a browser encounters a CSS selector or declaration it doesn't recognise.

The answer is that it does nothing, and just moves on to the next bit of CSS!

If a browser is parsing your rules, and encounters a property or value that it doesn't understand, it ignores it and moves on to the next declaration. It will do this if you have made an error and misspelled a property or value, or if the property or value is just too new and the browser doesn't yet support it.

Similarly, if a browser encounters a selector that it doesn't understand, it will just ignore the whole rule and move on to the next one.

In the example below I have used the British English spelling for color, which makes that property invalid as it is not recognised. So my paragraph has not been colored blue. All of the other CSS have been applied however; only the invalid line is ignored.

<p> I want this text to be large, bold and blue.</p>

p {

font-weight: bold;

colour: blue; /\* incorrect spelling of the color property \*/

font-size: 200%;

}

Open in CodePenOpen in JSFiddle

This behavior is very useful. It means that you can use new CSS as an enhancement, knowing that no error will occur if it is not understood — the browser will either get the new feature or not. Coupled with the way that the cascade works, and the fact that browsers will use the last CSS they come across in a stylesheet when you have two rules with the same specificity you can also offer alternatives for browsers that don't support new CSS.

This works particularly well when you want to use a value that is quite new and not supported everywhere. For example, some older browsers do not support calc() as a value. I might give a fallback width for a box in pixels, then go on to give a width with a calc() value of 100% - 50px. Old browsers will use the pixel version, ignoring the line about calc() as they don't understand it. New browsers will interpret the line using pixels, but then override it with the line using calc() as that line appears later in the cascade.

.box {

width: 500px;

width: calc(100% - 50px);

}

We will look at many more ways to support varying browsers in later lessons.

**CSS gradients** are represented by the [<gradient>](https://developer.mozilla.org/en-US/docs/Web/CSS/gradient) data type, a special type of [<image>](https://developer.mozilla.org/en-US/docs/Web/CSS/image) made of a progressive transition between two or more colors. You can choose between three types of gradients: *linear* (created with the [linear-gradient()](https://developer.mozilla.org/en-US/docs/Web/CSS/linear-gradient) function), *radial* (created with [radial-gradient()](https://developer.mozilla.org/en-US/docs/Web/CSS/radial-gradient)), and *conic* (created with the [conic-gradient()](https://developer.mozilla.org/en-US/docs/Web/CSS/conic-gradient) function). You can also create repeating gradients with the [repeating-linear-gradient()](https://developer.mozilla.org/en-US/docs/Web/CSS/repeating-linear-gradient), [repeating-radial-gradient()](https://developer.mozilla.org/en-US/docs/Web/CSS/repeating-radial-gradient), and [repeating-conic-gradient()](https://developer.mozilla.org/en-US/docs/Web/CSS/repeating-conic-gradient) functions.

Gradients can be used anywhere you would use an <image>, such as in backgrounds. Because gradients are dynamically generated, they can negate the need for the raster image files that traditionally were used to achieve similar effects. In addition, because gradients are generated by the browser, they look better than raster images when zoomed in, and can be resized on the fly.

We'll start by introducing linear gradients, then introduce features that are supported in all gradient types using linear gradients as the example, then move on to radial, conic and repeating gradients

**Using linear gradients**

A linear gradient creates a band of colors that progress in a straight line.

A basic linear gradient

To create the most basic type of gradient, all you need is to specify two colors. These are called *color stops*. You must have at least two, but you can have as many as you want.

.simple-linear {

background: linear-gradient(blue, pink);

}

Changing the direction

By default, linear gradients run from top to bottom. You can change their rotation by specifying a direction.

.horizontal-gradient {

background: linear-gradient(to right, blue, pink);

}

Diagonal gradients

You can even make the gradient run diagonally, from corner to corner.

.diagonal-gradient {

background: linear-gradient(to bottom right, blue, pink);

}

Using angles

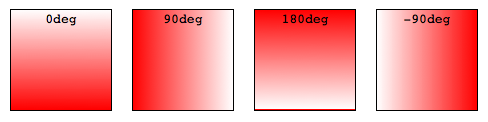
If you want more control over its direction, you can give the gradient a specific angle.

.angled-gradient {

background: linear-gradient(70deg, blue, pink);

}

When using an angle, 0deg creates a vertical gradient running bottom to top, 90deg creates a horizontal gradient running left to right, and so on in a clockwise direction. Negative angles run in the counterclockwise direction.



**Declaring colors & creating effects**

All CSS gradient types are a range of position-dependent colors. The colors produced by CSS gradients can vary continuously with position, producing smooth color transitions. It is also possible to create bands of solid colors, and hard transitions between two colors. The following are valid for all gradient functions:

Using more than two colors

You don't have to limit yourself to two colors—you may use as many as you like! By default, colors are evenly spaced along the gradient.

.auto-spaced-linear-gradient {

background: linear-gradient(red, yellow, blue, orange);

}

Positioning color stops

You don't have to leave your color stops at their default positions. To fine-tune their locations, you can give each one zero, one, or two percentage or, for radial and linear gradients, absolute length values. If you specify the location as a percentage, 0% represents the starting point, while 100% represents the ending point; however, you can use values outside that range if necessary to get the effect you want. If you leave a location unspecified, the position of that particular color stop will be automatically calculated for you, with the first color stop being at 0% and the last color stop being at 100%, and any other color stops being half way between their adjacent color stops.

.multicolor-linear {

background: linear-gradient(to left, lime 28px, red 77%, cyan);

}

Creating hard lines

To create a hard line between two colors, creating a stripe instead of a gradual transition, adjacent color stops can be set to the same location. In this example, the colors share a color stop at the 50% mark, halfway through the gradient:

.striped {

background: linear-gradient(to bottom left, cyan 50%, palegoldenrod 50%);

}

Gradient hints

By default, the gradient transitions evenly from one color to the next. You can include a color-hint to move the midpoint of the transition value to a certain point along the gradient. In this example, we've moved the midpoint of the transition from the 50% mark to the 10% mark.

.color-hint {

background: linear-gradient(blue, 10%, pink);

}

.simple-linear {

background: linear-gradient(blue, pink);

}

Creating color bands & stripes

To include a solid, non-transitioning color area within a gradient, include two positions for the color stop. Color stops can have two positions, which is equivalent to two consecutive color stops with the same color at different positions. The color will reach full saturation at the first color stop, maintain that saturation through to the second color stop, and transition to the adjacent color stop's color through the adjacent color stop's first position.

.multiposition-stops {

background: linear-gradient(to left,

lime 20%, red 30%, red 45%, cyan 55%, cyan 70%, yellow 80% );

background: linear-gradient(to left,

lime 20%, red 30% 45%, cyan 55% 70%, yellow 80% );

}

.multiposition-stop2 {

background: linear-gradient(to left,

lime 25%, red 25%, red 50%, cyan 50%, cyan 75%, yellow 75% );

background: linear-gradient(to left,

lime 25%, red 25% 50%, cyan 50% 75%, yellow 75% );

}

In the first example above, the lime goes from the 0% mark, which is implied, to the 20% mark, transitions from lime to red over the next 10% of the width of the gradient, reach solid red at the 30% mark, and staying solid red up until 45% through the gradient, where it fades to cyan, being fully cyan for 15% of the gradient, and so on.

In the second example, the second color stop for each color is at the same location as the first color stop for the adjacent color, creating a striped effect.

In both examples, the gradient is written twice: the first is the CSS Images Level 3 method of repeating the color for each stop and the second example is the CSS Images Level 4 multiple color stop method of including two color-stop-lengths in a linear-color-stop declaration.

Controlling the progression of a gradient

By default, a gradient evenly progresses between the colors of two adjacent color stops, with the midpoint between those two color stops being the midpoint color value. You can control the interpolation, or progression, between two color stops by including a color hint location. In this example, the color reaches the midpoint between lime and cyan 20% of the way through the gradient rather than 50% of the way through. The second example does not contain the hint to hilight the difference the color hint can make:

.colorhint-gradient {

background: linear-gradient(to top, black, 20%, cyan);

}

.regular-progression {

background: linear-gradient(to top, black, cyan);

}

Overlaying gradients

Gradients support transparency, so you can stack multiple backgrounds to achieve some pretty fancy effects. The backgrounds are stacked from top to bottom, with the first specified being on top.

.layered-image {

background: linear-gradient(to right, transparent, mistyrose),

url("https://mdn.mozillademos.org/files/15525/critters.png");

}

Stacked gradients

You can even stack gradients with other gradients. As long as the top gradients aren't entirely opaque, the gradients below will still be visible.

.stacked-linear {

background:

linear-gradient(217deg, rgba(255,0,0,.8), rgba(255,0,0,0) 70.71%),

linear-gradient(127deg, rgba(0,255,0,.8), rgba(0,255,0,0) 70.71%),

linear-gradient(336deg, rgba(0,0,255,.8), rgba(0,0,255,0) 70.71%);

}

**Using radial gradients**

Radial gradients are similar to linear gradients, except that they radiate out from a central point. You can dictate where that central point is. You can also make them circular or elliptical.

A basic radial gradient

As with linear gradients, all you need to create a radial gradient are two colors. By default, the center of the gradient is at the 50% 50% mark, and the gradient is elliptical matching the aspect ratio of it's box:

.simple-radial {

background: radial-gradient(red, blue);

}

Positioning radial color stops

Again like linear gradients, you can position each radial color stop with a percentage or absolute length.

.radial-gradient {

background: radial-gradient(red 10px, yellow 30%, #1e90ff 50%);

}

Positioning the center of the gradient

You can position the center of the gradient with keyterms, percentage, or absolute lengths, length and percentage values repeating if only one is present, otherwise in the order of position from the left and position from the top.

.radial-gradient {

background: radial-gradient(at 0% 30%, red 10px, yellow 30%, #1e90ff 50%);

}

Sizing radial gradients

Unlike linear gradients, you can specify the size of radial gradients. Possible values include closest-corner, closest-side, farthest-corner, and farthest-side, with farthest-corner being the default.

**Example: closest-side for ellipses**

This example uses the closest-side size value, which means the size is set by the distance from the starting point (the center) to the closest side of the enclosing box.

.radial-ellipse-side {

background: radial-gradient(ellipse closest-side,

red, yellow 10%, #1e90ff 50%, beige);

}

**Example: farthest-corner for ellipses**

This example is similar to the previous one, except that its size is specified as farthest-corner, which sets the size of the gradient by the distance from the starting point to the farthest corner of the enclosing box from the starting point.

.radial-ellipse-far {

background: radial-gradient(ellipse farthest-corner at 90% 90%,

red, yellow 10%, #1e90ff 50%, beige);

}

**Example: closest-side for circles**

This example uses closest-side, which makes the circle's size to be the distance between the starting point (the center) and the closest side. The circle's radius is the distance between the center of the gradient and the closest edge, which due to the positioning of the 25% from the top and 25% from the bottom, is closest to the bottom, since the height in this case is narrower than the width.

.radial-circle-close {

background: radial-gradient(circle closest-side at 25% 75%,

red, yellow 10%, #1e90ff 50%, beige);

}

Stacked radial gradients

Just like linear gradients, you can also stack radial gradients. The first specified is on top, the last on the bottom.

.stacked-radial {

background:

radial-gradient(circle at 50% 0,

rgba(255,0,0,.5),

rgba(255,0,0,0) 70.71%),

radial-gradient(circle at 6.7% 75%,

rgba(0,0,255,.5),

rgba(0,0,255,0) 70.71%),

radial-gradient(circle at 93.3% 75%,

rgba(0,255,0,.5),

rgba(0,255,0,0) 70.71%) beige;

border-radius: 50%;

}

**Using conic gradients**

The **conic-gradient()** [CSS](https://developer.mozilla.org/en-US/docs/Web/CSS) function creates an image consisting of a gradient with color transitions rotated around a center point (rather than radiating from the center). Example conic gradients include pie charts and color wheels, but they can also be used for creating checker boards and other intersting effects.

The conic-gradient syntax is similar to the radial-gradient syntax, but the color-stops are placed around a gradient arc, the circumference of a circle, rather than on the gradient line emerging from the center of the gradient, and the color-stops are percentages or degrees: absolute lengths are not valid.

In a radial gradient, the colors transition from the center of an ellipse, outward, in all directions. With conic gradients, the colors transition as as if spun around the center of a circle, starting at the top and going clockwise. Similar to radial gradients, you can position the center of the gradient. Similar to linear gradients, you can change the gradient angle.

A basic conic gradient

As with linear and radial gradients, all you need to create a conic gradient are two colors. By default, the center of the gradient is at the 50% 50% mark, with the start of the gradient facing up:

.simple-conic {

background: conic-gradient(red, blue);

}

Positioning the conic center

Like radial gradients, you can position the center of the conic gradient with keyterms, percentage, or absolute lengths, with the keyword "at"

.conic-gradient {

background: conic-gradient(at 0% 30%, red 10%, yellow 30%, #1e90ff 50%);

}

Changing the angle

Like radial gradients, you can position the center of the conic gradient with keyterms, percentage, or absolute lengths, with the keyword "at"

.conic-gradient {

background: conic-gradient(from 45deg, red, orange, yellow, green, blue, purple);

}

**Using repeating gradients**

The [linear-gradient()](https://developer.mozilla.org/en-US/docs/Web/CSS/linear-gradient), [radial-gradient()](https://developer.mozilla.org/en-US/docs/Web/CSS/radial-gradient), and [conic-gradient()](https://developer.mozilla.org/en-US/docs/Web/CSS/conic-gradient) functions don't support automatically repeated color stops. However, the [repeating-linear-gradient()](https://developer.mozilla.org/en-US/docs/Web/CSS/repeating-linear-gradient), [repeating-radial-gradient()](https://developer.mozilla.org/en-US/docs/Web/CSS/repeating-radial-gradient), and [repeating-conic-gradient()](https://developer.mozilla.org/en-US/docs/Web/CSS/repeating-conic-gradient) functions are available to offer this functionality.

The size of the gradient line or arc that repeats is the length between the first color stop value and the last color stop length value. If the first color stop just has a color and no color stop length, the value defaults to 0. If the last color stop has just a color and no color stop length, the value defaults to 100%. If neither is declared, the gradient line is 100% meaning the linear and conic gradients will not repeat and the radial gradient will only repeat if the radius of the gradient is smaller than the length between the center of the gradient and the farthest corner. If the first color stop is declared, and the value is greater than 0, the gradient will repeat, as the size of the line or arc is the difference between the first color stop and last color stop is less than 100% or 360 degrees.

Repeating linear gradients

This example uses [repeating-linear-gradient()](https://developer.mozilla.org/en-US/docs/Web/CSS/repeating-linear-gradient) to create a gradient that progresses repeatedly in a straight line. The colors get cycled over again as the gradient repeats. In this case the gradient line is 10px long.

.repeating-linear {

background: repeating-linear-gradient(-45deg, red, red 5px, blue 5px, blue 10px);

}

Multiple repeating linear gradients

Similar to regular linear and radial gradients, you can include multiple gradients, one on top of the other. This only makes sense if the gradients are partially transparent allowing subsequent gradients to show through the transparent areas, or if you include different [background-sizes](https://developer.mozilla.org/en-US/docs/Web/CSS/background-size), optionally with different [background-position](https://developer.mozilla.org/en-US/docs/Web/CSS/background-position) property values, for each gradient image. We are using transparency.

In this case the gradient lines are 300px, 230px, and 300px long.

.multi-repeating-linear {

background:

repeating-linear-gradient(190deg, rgba(255, 0, 0, 0.5) 40px,

rgba(255, 153, 0, 0.5) 80px, rgba(255, 255, 0, 0.5) 120px,

rgba(0, 255, 0, 0.5) 160px, rgba(0, 0, 255, 0.5) 200px,

rgba(75, 0, 130, 0.5) 240px, rgba(238, 130, 238, 0.5) 280px,

rgba(255, 0, 0, 0.5) 300px),

repeating-linear-gradient(-190deg, rgba(255, 0, 0, 0.5) 30px,

rgba(255, 153, 0, 0.5) 60px, rgba(255, 255, 0, 0.5) 90px,

rgba(0, 255, 0, 0.5) 120px, rgba(0, 0, 255, 0.5) 150px,

rgba(75, 0, 130, 0.5) 180px, rgba(238, 130, 238, 0.5) 210px,

rgba(255, 0, 0, 0.5) 230px),

repeating-linear-gradient(23deg, red 50px, orange 100px,

yellow 150px, green 200px, blue 250px,

indigo 300px, violet 350px, red 370px);

}

Plaid gradient

To create plaid we include several overlapping gradients with transparency. In the first background declaration we listed every color stop separately. The second background property declaration using the multiple position color stop syntax:

.plaid-gradient {

background:

repeating-linear-gradient(90deg, transparent, transparent 50px,

rgba(255, 127, 0, 0.25) 50px, rgba(255, 127, 0, 0.25) 56px,

transparent 56px, transparent 63px,

rgba(255, 127, 0, 0.25) 63px, rgba(255, 127, 0, 0.25) 69px,

transparent 69px, transparent 116px,

rgba(255, 206, 0, 0.25) 116px, rgba(255, 206, 0, 0.25) 166px),

repeating-linear-gradient(0deg, transparent, transparent 50px,

rgba(255, 127, 0, 0.25) 50px, rgba(255, 127, 0, 0.25) 56px,

transparent 56px, transparent 63px,

rgba(255, 127, 0, 0.25) 63px, rgba(255, 127, 0, 0.25) 69px,

transparent 69px, transparent 116px,

rgba(255, 206, 0, 0.25) 116px, rgba(255, 206, 0, 0.25) 166px),

repeating-linear-gradient(-45deg, transparent, transparent 5px,

rgba(143, 77, 63, 0.25) 5px, rgba(143, 77, 63, 0.25) 10px),

repeating-linear-gradient(45deg, transparent, transparent 5px,

rgba(143, 77, 63, 0.25) 5px, rgba(143, 77, 63, 0.25) 10px);

background:

repeating-linear-gradient(90deg, transparent 0 50px,

rgba(255, 127, 0, 0.25) 50px 56px,

transparent 56px 63px,

rgba(255, 127, 0, 0.25) 63px 69px,

transparent 69px 116px,

rgba(255, 206, 0, 0.25) 116px 166px),

repeating-linear-gradient(0deg, transparent 0 50px,

rgba(255, 127, 0, 0.25) 50px 56px,

transparent 56px 63px,

rgba(255, 127, 0, 0.25) 63px 69px,

transparent 69px 116px,

rgba(255, 206, 0, 0.25) 116px 166px),

repeating-linear-gradient(-45deg, transparent 0 5px,

rgba(143, 77, 63, 0.25) 5px 10px),

repeating-linear-gradient(45deg, transparent 0 5px,

rgba(143, 77, 63, 0.25) 5px 10px);

}

Repeating radial gradients

This example uses [repeating-radial-gradient()](https://developer.mozilla.org/en-US/docs/Web/CSS/repeating-radial-gradient) to create a gradient that radiates repeatedly from a central point. The colors get cycled over and over as the gradient repeats.

.repeating-radial {

background: repeating-radial-gradient(black, black 5px, white 5px, white 10px);

}

Multiple repeating radial gradients

.multi-target {

background:

repeating-radial-gradient(ellipse at 80% 50%,rgba(0,0,0,0.5),

rgba(0,0,0,0.5) 15px, rgba(255,255,255,0.5) 15px,

rgba(255,255,255,0.5) 30px) top left no-repeat,

repeating-radial-gradient(ellipse at 20% 50%,rgba(0,0,0,0.5),

rgba(0,0,0,0.5) 10px, rgba(255,255,255,0.5) 10px,

rgba(255,255,255,0.5) 20px) top left no-repeat yellow;

background-size: 200px 200px, 150px 150px;

}