

# CSS Tutorial

# Contents of CSS Course

**Applying CSS** – The different ways you can apply CSS to HTML.

**Selectors, Properties, and Values** – The bits that make up CSS.

**Colors** – How to use color.

**Text** – How to manipulate the size and shape of text.

**Margins and Padding** – How to space things out.

**Borders** – Erm. Borders. Things that go around things.

**Class and ID Selectors**: Make your own selectors without the need for sticky-backed plastic!

**Specificity**: How a browser will deal with conflicting CSS rules.

**Page Layout**: Floating and positioning boxes.



# Contents of CSS Course

**Rounded Corners:** Corners. That are rounded.

**Shadows:** Adding “pop” to boxes and text.

aim with clever selectors.

**Advanced Colors:** Alpha transparency and HSL.

**At-Rules:** Importing style sheets, styles for different **media types**, specifying the character set of a style sheet and **embedded fonts**.

**Attribute Selectors:** Targeting boxes by their elements' HTML attributes.

**CSS Transitions:** Creating smooth animations.

**Backgrounds:** Multiples, Size, and Origin

**Transformations:** Molding the size and shape of a box and its contents.

**Gradients:** Linear and radial gradients without image files.

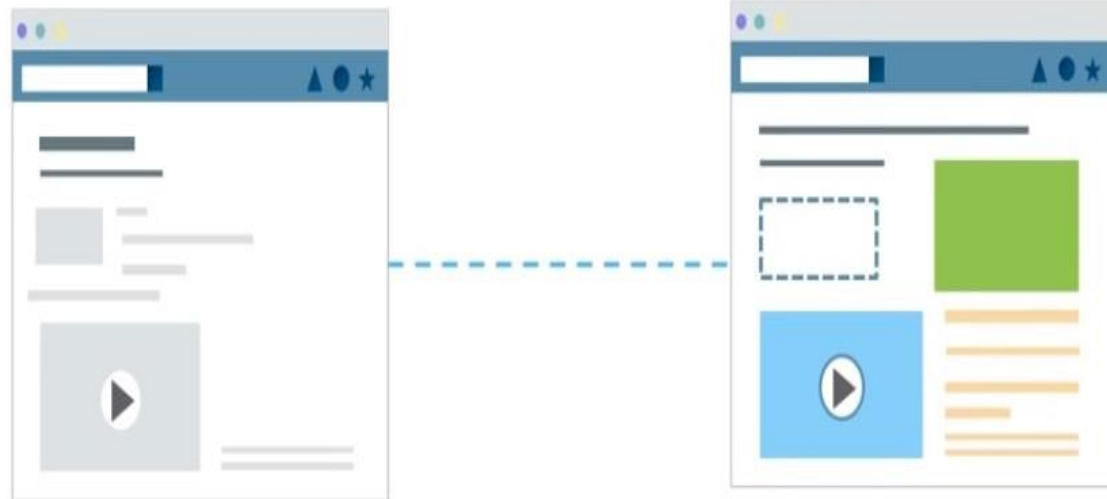
**Media Queries:** Optimizing pages for different devices and screen sizes.

# What Is Cascading Style Sheets?

---



- CSS describes how HTML elements are displayed
- Used to style the web



# HTML + CSS

---

HTML



CSS



# What is CSS?



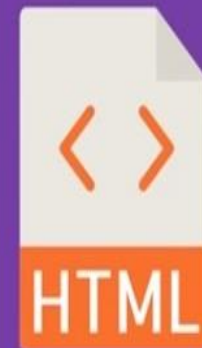
Cascading Style Sheet



Controls how HTML is  
Displayed



Developed and  
Maintained by W3C



Solved problems in  
HTML3.2

# What is css?

CSS stands for **Cascading Style Sheets**

CSS describes how HTML elements are to be displayed on screen, paper, or in other media

CSS saves a lot of work. It can control the layout of multiple web pages all at once

External stylesheets are stored in CSS files

## Why Use CSS?

CSS is used to define styles for your web pages, including the design, layout and variations in display for different devices and screen sizes.

## CSS Syntax

A CSS rule-set consists of a selector and a declaration block:





# CSS Syntax



## Basic CSS Syntax

```
Selector{  
  property1: value1;  
  property2: value2;  
}
```

## Including CSS to your HTML

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

## Inline CSS

```
<tag style="property: value"></tag>
```

## Internal CSS

```
<style>  
  selector {  
    property : value  
  }  
</style>
```



# Applying CSS

- ▶ There are three ways to apply CSS to HTML: **Inline**, **internal**, and **external**.
- ▶ **Inline**
- ▶ Inline styles are plunked straight into the HTML tags using the style attribute.
- ▶ They look something like this:
- ▶ `<p style="color: red">text</p>` This will make that specific paragraph red.

# Applying CSS

## Internal

Embedded, or internal, styles are used for the whole page.

Inside the head element, the style tags surround all of the styles for the page.

```
<!DOCTYPE html>
<html>
<head>
<title>CSS Example</title>
<style>
  p { color: red; }
  a { color: blue; }
</style>
</head>
```

# Applying CSS

- ▶ **External**

- ▶ External styles are used for the whole, multiple-page website. There is a **separate CSS file**, which will simply look something like:

```
p { color: red; }
```

```
a { color: blue; }
```

If this file is saved as “style.css” in the same directory as your HTML page then it can be linked to in the HTML like this:

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<title>CSS Example</title>
```

```
<link rel="stylesheet" href="style.css"> ...
```

# CSS Selector

- ▶ Whereas HTML has **tags**, CSS has **selectors**. Selectors are the names given to styles in internal and external style sheets.
- ▶ For each selector there are “**properties**” inside **curly brackets**, which simply take the form of words such as color, font-weight or background-color.
- ▶ A **value** is given to the property following a **colon** (NOT an “equals” sign). **Semi-colons** are used to separate the properties.
- ▶ `body { font-size: 14px; color: navy; }` This will apply the given values to the font-size and color properties to the body selector.

# CSS Selectors

CSS



HTML

Selector	Use
*	All elements
div	All divs
div, p	All divs and paragraphs
div p	All paragraphs inside divs
div > p	<i>all p tags, one level deep in div</i>
div + p	<i>p tags immediately after div</i>
div ~ p	<i>p tags preceded by div</i>
.classname	<i>all elements with class</i>
#idname	<i>element with ID</i>
div.Classname	<i>divs with certain classname</i>
div#idname	<i>div with certain ID</i>
#idname *	<i>all elements inside #idname</i>

# CSS Selectors

CSS



HTML

Pseudo Selector	Use
A:hover	All elements
A:active	All divs
A:visited	All divs and paragraphs
Div:empty	All paragraphs inside divs
P:first-of-type	<i>all p tags, one level deep in div</i>
P:last-of-type	<i>p tags immediately after div</i>
P:first-child	<i>p tags preceded by div</i>
P:nth-child(2)	<i>all elements with class</i>

Attribute Selector	Use
a[target]	<i>links with a target attribute</i>
a[target="_blank"]	<i>links which open in new tab</i>
[title~="chair"]	<i>title element containing a word</i>
[class^="chair"]	<i>class starts with chair</i>
[class]="chair"]	<i>class starts with the chair word</i>
[class*="chair"]	<i>class contains chair</i>
[class\$="chair"]	<i>class ends with chair</i>
input[type="button"]	<i>specified input type</i>



# CSS Units

## Absolute Lengths

The **absolute length** units are **fixed** and a length expressed in any of these will **appear as exactly that size**.

Unit	Description
cm	centimeters
mm	millimeters
in	inches (1in = 96px = 2.54cm)
px *	pixels (1px = 1/96th of 1in)
pt	points (1pt = 1/72 of 1in)
pc	picas (1pc = 12 pt)

\* **Pixels (px)** are **relative to the viewing device**. For **low-dpi** devices, 1px is one device pixel (dot) of the display. For printers and high resolution screens 1px implies multiple device pixels.



# Relative Lengths

## Relative Lengths

Relative length units specify a length relative to another length property.

Unit	Description
em	Relative to the font-size of the element (2em means 2 times the size of the current font)
ex	Relative to the x-height of the current font (rarely used)
ch	Relative to width of the "0" (zero)
rem	Relative to font-size of the root element
vw	Relative to 1% of the width of the viewport
vh	Relative to 1% of the height of the viewport
vmin	Relative to 1% of viewport's smaller dimension
vmax	Relative to 1% of viewport's larger dimension
%	Relative to the parent element

# CSS Units

## Pixel

Represents pixel on a device

## Points

72 points equals 1 inch

## Ems

1 em = current font size

## Percent

Current font size is 100%



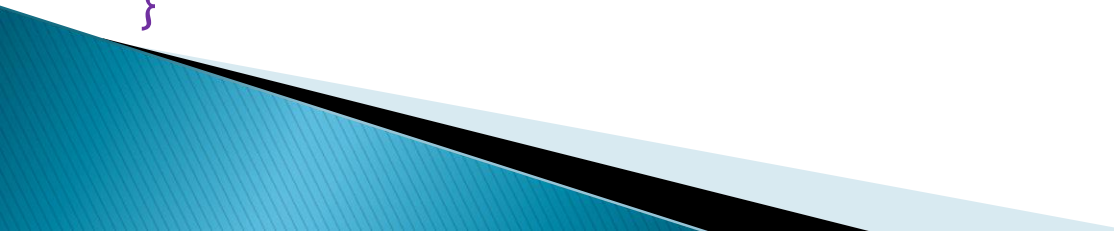
# Colors

## color and background-color

Colors can be applied by using **color** and **background-color** (note that this must be the American English “color” and not “colour”).

A blue background and yellow text could look like this:

```
h1 { color: yellow;  
background-color: blue;  
}  
body { font-size: 14px;  
color: navy;  
}  
h2 { color: #ffc;  
background-color: #009;  
}
```



# CSS Color

CSS brings 16,777,216 colors to your disposal. They can take the form of a **name**, an **RGB**(red/green/blue) value or a **hex** code.

The following values, to specify full-on as red-as-red-can-be, all produce the same result:

red

rgb(255,0,0)

rgb(100%,0%,0%)

#ff0000

#f00

**Predefined color names**

include aqua, black, blue, fuchsia, gray, green, lime, maroon, avy, olive, purple, red, silver, teal, white, and yellow. transparent is also a valid value.

# CSS Text

## font-family

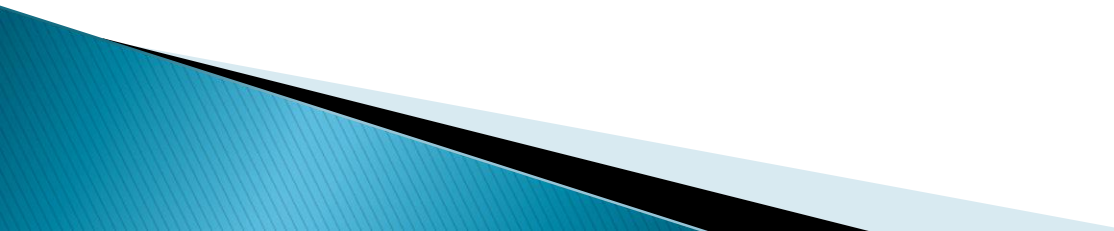
We use font-family to change font style of HTML elements.

Example “

```
Font-family: font-family:'BankGothic Lt BT';
```

```
font-family:'Lucida Calligraphy';
```

This is useful because different computers sometimes have different fonts installed. So font-family: arial, helvetica, serif, will look for the Arial font first and, if the browser can't find it, it will search for Helvetica, and then a common serif font.



# CSS Text

## font-size

**font-size** sets the size of the font.

## font-weight

Most commonly this is used as **font-weight: bold** or **font-weight: normal** but other values are **bolder**, **lighter**

## font-style

**font-style** states whether the text is italic or not. It can be **font-style: italic** or **font-style: normal**.

## text-decoration

**text-decoration** states whether the text has got a line running under, over, or through it.

**text-decoration: underline**, does what you would expect.

**text-decoration: overlie** places a line above the text.

**text-decoration: line-through** puts a line through the text (“strike-through”).

# CSS Text

## text-transform

**text-transform** will change the case of the text.

**text-transform: capitalize** turns the first letter of every word into uppercase.

**text-transform: uppercase** turns everything into uppercase.

**text-transform: lowercase** turns everything into lowercase.

**text-transform: none** I'll leave for you to work out.

## Text spacing

The **letter-spacing** and **word-spacing** properties are for spacing between letters or words. The value can be a length or normal.

The **line-height** property sets the height of the lines in an Element.

The **text-align** property will align the text inside an element to left, right, center, or justify.





# CSS Properties

```
body { font-family: arial, helvetica, sans-serif; font-size: 14px;
}
```

```
h1 { font-size: 2em; }
```

```
h2 { font-size: 1.5em; }
```

```
a { text-decoration: none; }
```

```
strong { font-style: italic; text-transform: uppercase;
}
```

```
p { letter-spacing: 0.5em;
```

```
word-spacing: 2em;
```

```
line-height: 1.5;
```

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



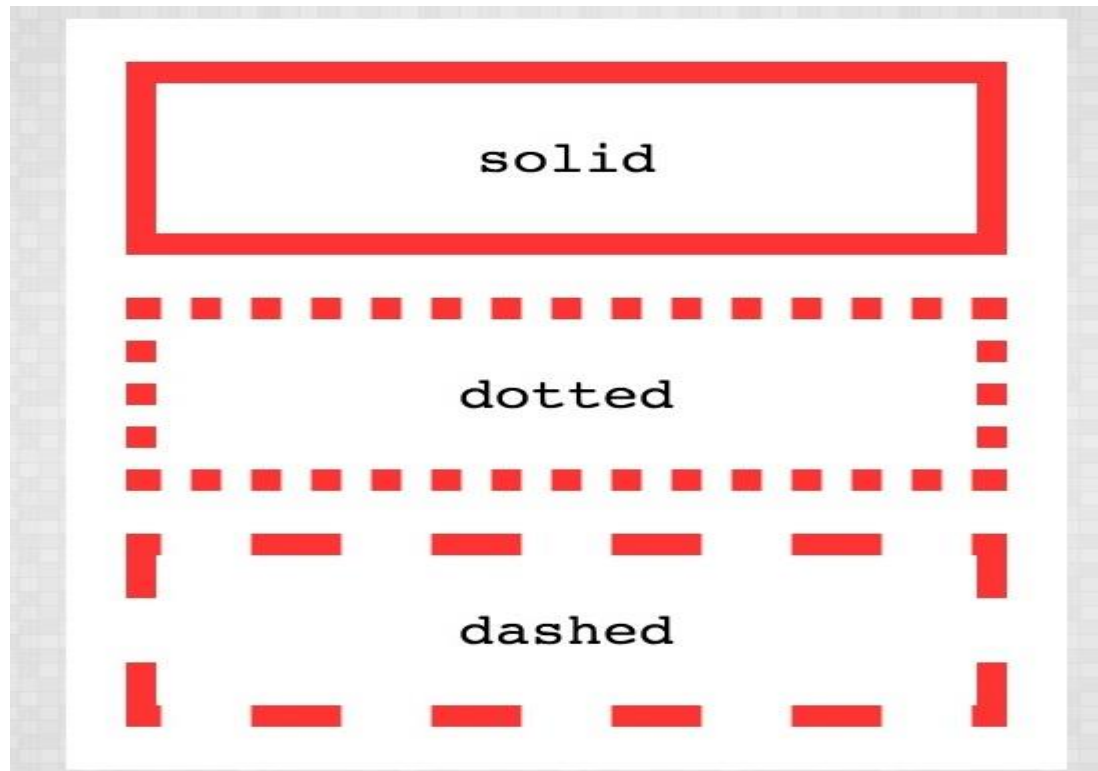
# Margin and Padding

A margin is the space **outside** something, whereas padding is the space **inside** something.

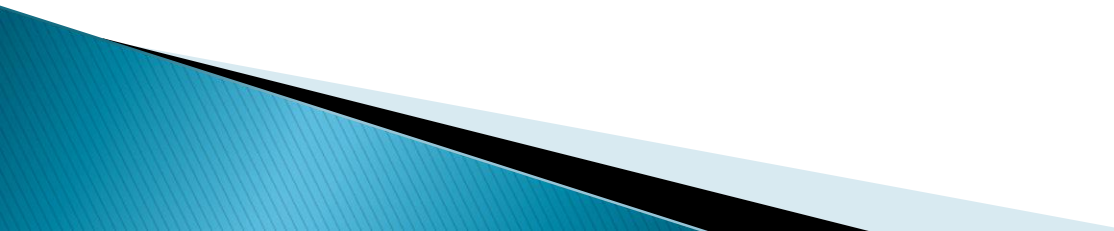


# Borders

- ▶ To make a border around an element, all you need is **border-style**. The values can
- ▶ be solid, dotted, dashed, double, groove, ridge, inset and outset.



# ID and class

- ▶ You can also define your own selectors in the form of **class** and **ID** selectors.
  - ▶ The benefit of this is that you can have the same HTML element, but present it differently depending on its class or ID.
  - ▶ In the CSS, a class selector is a name preceded by a **full stop** (".") and an ID selector is a name preceded by a **hash character** ("#").
- 



# Class and ID Selector

In the CSS, a class selector is a name preceded by a **full stop** (“.”) and an ID selector is a name preceded by a **hash character** (“#”).

```
<div id="top"> <h1>Chocolate curry</h1>
```

```
<p class="intro">This is my recipe for making curry purely with  
chocolate</p>
```

```
<p class="intro">Mmm mm mmmmm</p> </div>
```

So the CSS might look something like:

```
#top { background-color: #ccc;  
padding: 20px }
```

```
.intro { color: red; font-weight: bold; }
```

The difference between an ID and a class is that an ID can be used to identify **one element**, whereas a class can be used to identify **more than one**.

# Grouping and Nesting

You can give the same properties to a number of selectors without having to repeat them.

For example, if you have something like:

```
h2, .thisOtherClass, .yetAnotherClass {  
  color: red; }
```

## Nesting

If the CSS is structured well, there shouldn't be a need to use many class or ID selectors. This is because you can specify properties to selectors **within** other selectors.

For example:

```
#top {  
  background-color: #ccc;  
  padding: 1em }  
#top h1 { color: #ff0; }  
#top p { color: red;  
  font-weight: bold; }
```



# Pseudo Classes

## Dynamic Pseudo Classes

**active** is for when something activated by the user, such as when a link is clicked on.

**hover** is for a when something is passed over by an input from the user, such as when a cursor moves over a link.

**focus** is for when something gains focus, that is when it is selected by, or is ready for, keyboard input.

```
a:active { color: red; }  
a:hover {  
text-decoration: none;  
color: blue;  
background-color: yellow;  
}  
input:focus, textarea:focus {  
background: #eee;  
}
```

# First Child

- ▶ There is the first-child pseudo class. This will target something only if it is the very first descendant of an element.

<body>

<p>I'm the body's first paragraph child. I rule. Obey me.</p>

<p>I resent you.</p> ...

...if you only want to style the **first** paragraph, you could use the following CSS:

```
p:first-child {  
  font-weight: bold; font-size: 40px; }
```

# Background Image

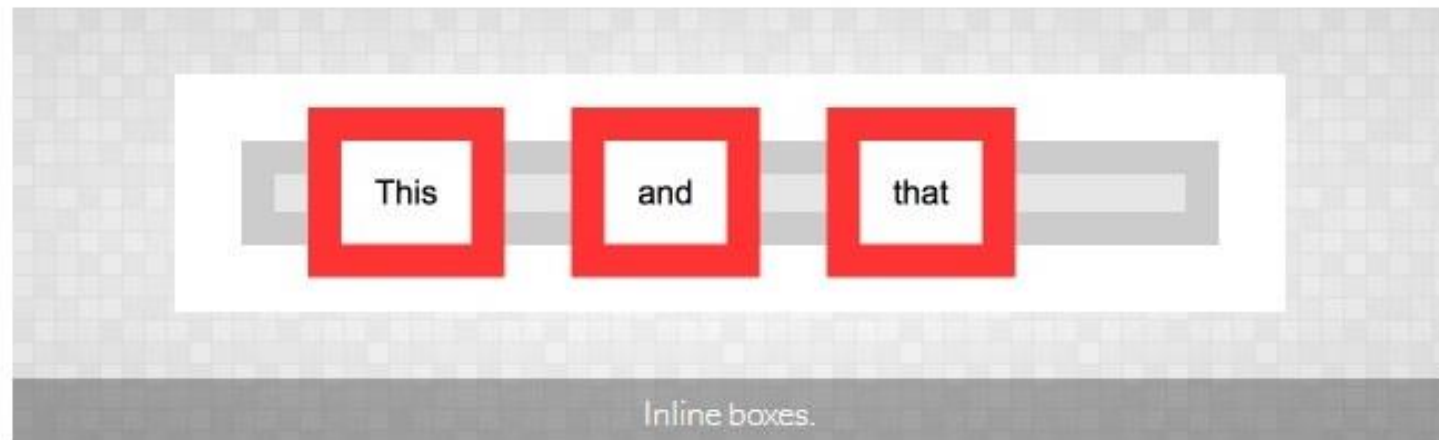
- ▶ CSS background images are a powerful way to add detailed presentation to a pag
- ▶ background-image, which is the location of the image itself.
- ▶ background-repeat, which is how the image repeats itself. Its value can be:
  - **repeat**, the equivalent of a “tile” effect across the whole background,
  - **repeat-y**, repeating on the y-axis, above and below,
  - **repeat-x** (repeating on the x-axis, side-by-side), or
  - **no-repeat** (which shows just one instance of the image).
- ▶ background-position, which can be top, center, bottom, left, right, a length, or a percentage, or any sensible combination, such as top right.

```
body { background: white  
        url(http://www.htmldog.com/images/bg.gif) no-repeat top  
        right; }
```



# Display

- ▶ The most fundamental types of display
- ▶ are **inline**, **block** and **none** and they can be manipulated with the display property and the shockingly surprising values inline, block and none.
- ▶ **Inline**
- ▶ inline does just what it says — boxes that are displayed inline follow the flow of a line. **Anchor (links)** and **emphasis** are examples of elements that are displayed inline by default.
- ▶ `li { display: inline }`



# Display

## Block

- ▶ block makes a box standalone, fitting the entire width of its containing box, with an effective line break before and after it. Unlike inline boxes, block boxes allow greater manipulation of height, margins, and padding. **Heading and paragraph elements are examples** of elements that are displayed this way by default in browsers.
- ▶ `#navigation a { display: block; padding: 20px 10px; }`



# Display

- ▶ **None**
- ▶ none, well, **doesn't display a box at all**, it can be used to good effect with dynamic effects, such as switching extended information on and off at the click of a link, or in alternative stylesheets.

```
#navigation, #related_links { display: none }
```



# Pseudo Elements

- ▶ **First Letters and First Lines**
- ▶ The first-letter pseudo element applies to the first letter inside a box and first-line to the top-most displayed line in a box.

```
p { font-size: 12px; }  
p:first-letter {  
  font-size: 24px;  
  float: left; }  
p:first-line {  
  font-weight: bold;  
}
```

As an example, you could create drop caps and a bold first-line for paragraphs with something like this:

---

**O**nce upon a time  
in a blueberry bubblegum land  
covered in pink violets that swayed  
to the rhythm of "My Baby Just  
Cares for Me" there lived a podgy yet attractive  
raspberry fairy called Bedooda.

---

# Pseudo Element

The before and after pseudo elements are used in conjunction with the content property to place content either side of a box without touching the HTML.

```
blockquote:before {  
  content: open-quote;  
}
```

```
blockquote:after {  
  content: close-quote;  
}
```

```
li:before {  
  content: "POW! ";  
}
```

```
p:before {  
  content: url(images/jam.jpg);  
}
```

# Pseudo Element

- ▶ The content property effectively creates another box to play with so you can also add styles to the “presentational content”:

```
li:before {  
  content: "POW! ";  
  background: red;  
  color: #fc0;  
}
```

# Positioning

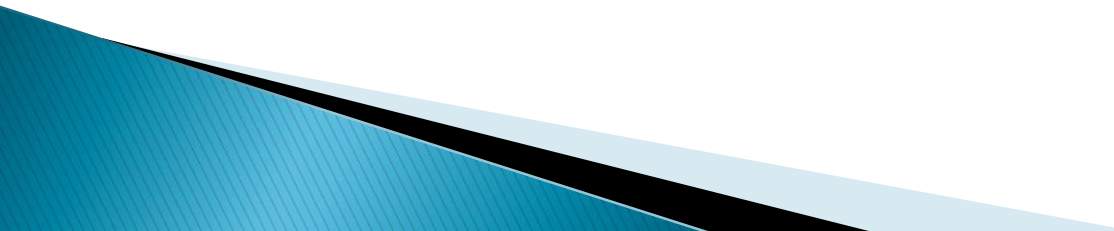
The **position** property is used to define whether a box is **absolute**, **relative**, **static** or **fixed**:

**static** is the **default** value and renders a box in the normal order of things, as they appear in the HTML.

**relative** is much like static but the **box can be offset from its original position** with the properties **top**, **right**, **bottom** and **left**.

**absolute** pulls a box out of the normal flow of the HTML and delivers it to a world all of its own. In this crazy little world, the **absolute box can be placed anywhere on the page** using **top**, **right**, **bottom** and **left**.

**fixed** behaves like absolute, but it will absolutely position a box in reference to the browser window as opposed to the web page, so **fixed boxes should stay exactly where they are on the screen even when the page is scrolled**.



# Positioning

- ▶ Layout using absolute positioning

```
<div id="navigation">
```

```
<ul>
```

```
<li><a href="this.html">This</a></li>
```

```
<li><a href="that.html">That</a></li>
```

```
<li><a href="theOther.html">The Other</a></li>
```

```
</ul> </div> <div id="content">
```

```
<h1>Ra ra banjo banjo</h1>
```

```
<p>Welcome to the Ra ra banjo banjo page. Ra ra banjo banjo. Ra ra  
banjo banjo. Ra ra banjo banjo.</p>
```

```
<p>(Ra ra banjo banjo)</p>
```

```
</div>
```



# Floating

Floating a box will shift it to the right or left of a line, with surrounding content flowing around it.

Using float, you can float: left or float: right.

```
#navigation {
```

```
  float: left;
```

```
  width: 200px; }
```

```
#navigation2 {
```

```
  float: right;
```

```
  width: 200px; }
```

```
#content { margin: 0 200px; }
```



# Clear

- ▶ Then, if you do not want the next box to wrap around the floating objects, you can apply the clear property:
- ▶ **clear: left** will clear left floated boxes
- ▶ **clear: right** will clear right floated boxes
- ▶ **clear: both** will clear both left and right floated boxes.

```
<div id="footer">  
  <p>Footer! Hoorah!</p>  
</div>
```

...and then add the following CSS:

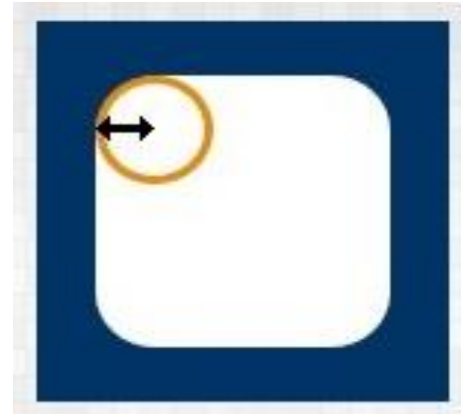
```
#footer { clear: both; }
```

# Rounded Corner

## Border radius?

The `border-radius` property can be used to add a corner to each corner of a box.

```
#marilyn {  
  background: #fff;  
  width: 100px;  
  height: 100px;  
  border-radius: 20px;  
}
```



## Ellipses

- ▶ You can specify different horizontal and vertical radii by splitting values with a “/”.
- ▶ #nanoo {  
 background: #fff;  
 width: 100px; height: 150px;  
 border-radius: 50px/100px;  
 border-bottom-left-radius: 50px;  
 border-bottom-right-radius: 50px;  
}



# Shadow

## Box Shadows

**box-shadow** is the standard CSS property to get you going and it can have a value comprising several parts:

**box-shadow: 5px 5px 3px 1px #999**

The first value is the **horizontal offset** — how far the shadow is nudged to the right (or left if it's negative)

The second value is the **vertical offset** — how far the shadow is nudged downwards (or upwards if it's negative)

The third value is the **blur radius** — the higher the value the less sharp the shadow. ("0" being absolutely sharp). This is optional — omitting it is equivalent of setting "0".

The fourth value is the **spread distance** — the higher the value, the larger the shadow ("0" being the inherited size of the box).

This is also optional — omitting it is equivalent of setting "0".

The fifth value is a **color**. That's optional, too.

# Shadow

## Inner shadows

You can also apply shadows to the inside of a box by adding “inset” to the list:

```
box-shadow: inset 0 0 7px 5px #ddd;
```



## Text Shadows

you can also apply shadows to the outline of text with **text-shadow**:

```
text-shadow: -2px 2px 2px #999;
```

The first value is the **horizontal offset**

The second value is the **vertical offset**

The third value is the **blur radius** (optional)

The fourth value is the **color** (optional, although omitting this will make the shadow the same color as the text itself)

# Universal selectors

Using an **asterisk** (“ \* ”), you can target **everything** under the sun. You can use it by itself to set global styles for a page, or as a descendant of a selector to set styles of everything within something.

The following, for example, will set the margin and padding on everything in a page to zero and everything within an element with the ID “contact” to be displayed as a block:

```
{ margin: 0;  
padding: 0;  
}  
  
#contact * {  
display: block;  
}
```

# Advanced Color

- ▶ Colors can be defined by **name**, **RGB**, or **hex values**, but **CSS 3** also allows you to paint away with **HSL** — hue, saturation, and **lightness** — as well as stipulating **transparency**.

**HSL** and **RGBA** (the “a” standing for “alpha”, as in “alpha transparency”) can be applied to any property that has a **color value**, such as **color**, **background-color**, **border-color** or **box-shadow**, to name a mere handful.

# RBGa Alpha Transparency

```
h1 { padding: 50px;  
background-image: url(snazzy.jpg);  
color: rgba(0,0,0,0.8); }
```

- ▶ A standard value of `rgb(0,0,0)` would set the heading to pure black but that fourth value, in `rgba`, sets the level of transparency, “1” being completely opaque, “0” being completely transparent. So `rgba(0,0,0,0.8)` is saying red=“0”, green=“0”, blue=“0”, alpha=“0.8”, which, all together, makes it 80% black.
- ▶ This doesn't only apply to text, you could apply a transparent background color to an entire box, a transparent box shadow... anywhere where you can use `rgb`, you can use `rgba`.

▶

# Hue saturation and lightness

- ▶ It is used in a similar way to rgb:
- ▶ `#smut { color: hsl(36, 100%, 50%) }`
- ▶ Rather than each sub-value being a part of the color spectrum, however, they are:
- ▶ **Hue** (“36” in the above example): Any angle, from 0 to 360, taken from a typical color wheel, where “0” (and “360”) is red, “120” is green and “240” is blue.
- ▶ **Saturation** (“100%” in the example): How saturated you want the color to be, from 0% (none, so a level of grey depending on the lightness) to 100% (the whole whack, please).
- ▶ **Lightness** (“50%” in the example): From 0% (black) to 100% (white), 50% being “normal”.
- ▶ So the example used here will produce an orange (36°) that is rich (100% saturation) and vibrant (50% lightness). It is the equivalent of #ff9900, #f90, and rgb(255, 153, 0).
- ▶ Here’s HSLa:
- ▶ `#rabbit { background: hsla(0, 75%, 75%, 0.5) }`

# At-Rules @import and @font-face

## Importing

- ▶ This is used to attach another stylesheet onto your existing one.
- ▶ `@import url(morestyles.css);`
- ▶ This can be used if a site requires long, complex stylesheets that might be easier to manage if they are broken down into smaller files.
- ▶ @import rules must be placed at the top of a stylesheet, before any other rules.

# Embedding fonts @font-face

- ▶ **@font-face** is now widely used as a technique for **embedding fonts in a web page** so that a typeface can be used even if it **isn't sitting on the user's computer**. So you no longer need to rely on “web safe” fonts such as Arial or Verdana.

```
@font-face {
```

```
font-family: "font of all knowledge";
```

```
src: url(fontofallknowledge.woff); }
```

- ▶ What this does is create a font named “font of all knowledge” using the font-family **descriptor** and **links the font file** “fontofallknowledge.woff” to that name using the src descriptor. “font of all knowledge” can then be used in a standard font rule, such as:

```
p {
```

```
font-family: "font of all knowledge",
```

```
arial, sans-serif; }
```



# CSS Transitions

- ▶ **Transitions** allow you to easily **animate** parts of your design without the need for the likes of JavaScript.
- ▶ Think of a traditional **:hover state**, where you might **change the appearance of a link** when a cursor fondles it:
- ▶ `a:link { color: hsl(36,50%,50%); }`
- ▶ `a:hover { color: hsl(36,100%,50%); }`

**The transition property**, allowing **smooth animation** (rather than a jump from one state to another).

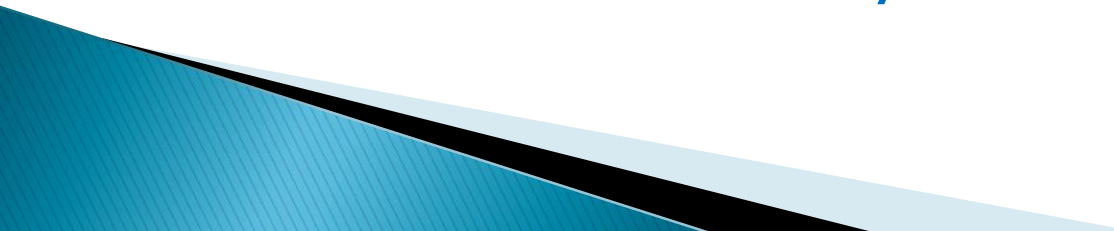
```
a:link
{ transition: all .5s linear 0;
  color: hsl(36,50%,50%); }
a:hover {
  color: hsl(36,100%,50%);
}
```

# CSS Transitions

- **transition-property**: which property (or properties) will transition.
- **transition-duration**: how long the transition takes.
- **transition-timing-function**: if the transition takes place at a constant speed or if it accelerates and decelerates.
- **transition-delay**: how long to wait until the transition takes place.

**transition: all .5s linear 0;**

...when the link is hovered over, the color will gradually change rather than suddenly switch. The **transition property** here is saying it wants **all properties** transitioned **over half a second** in **a linear fashion** with **no delay**.



# Targeting specific properties

While “all” can be used in transition–property (or transition), you can tell a browser only to transition certain properties, by simply **mentioning the property name** you want to change. So the previous example could actually include

transition: **color** .5s ease 0, given only the color changes.

If you want to target **more than one property** (without using “all”), you can offer up a **comma-separated list with transition property**:

```
a:link { transition: .5s; transition-property: color, font-size; ...
```

Or you can offer up **a slew of rules for transitioning each property** like so:

```
a:link { transition: color .5s, font-size 2s; ...
```



# Multiple backgrounds

you can also apply **multiple backgrounds**, adjust the **size of background images**, and define the **origin of a background** based on levels of the box model


CSS3 allows you to apply multiple background images to a single box by simply putting image locations in a comma separated list:

```
background-image: url(this.jpg), url(that.gif), url(theother.png);
```

You can also define all of the other background aspects

```
background: url(bg.png), url(bullet.png) 0 50% no-repeat,  
            url(arrow.png) right no-repeat;
```

It's just the same as using **background-image**, **background-position**, **background-repeat**, **background-attachment**



# Background Size

The background-size property allows you to stretch or compress a background image.

**auto**, which maintains the background image's original size and width/height ratio.

**lengths**, a width and a height, such as 100px 50px Specifying singlelength, such as 100px will result the equivalent of 100px auto.

**percentages**, a width and a height, such as 50% 25%

**contain**, which maintains the background image's original ratio and makes it as large as possible whilst fitting entirely within the box's background area.

**cover**, which maintains the background image's original ratio and makes it large enough to fill the entire background area, which may result in cropping of either the height or width.

# Transformation

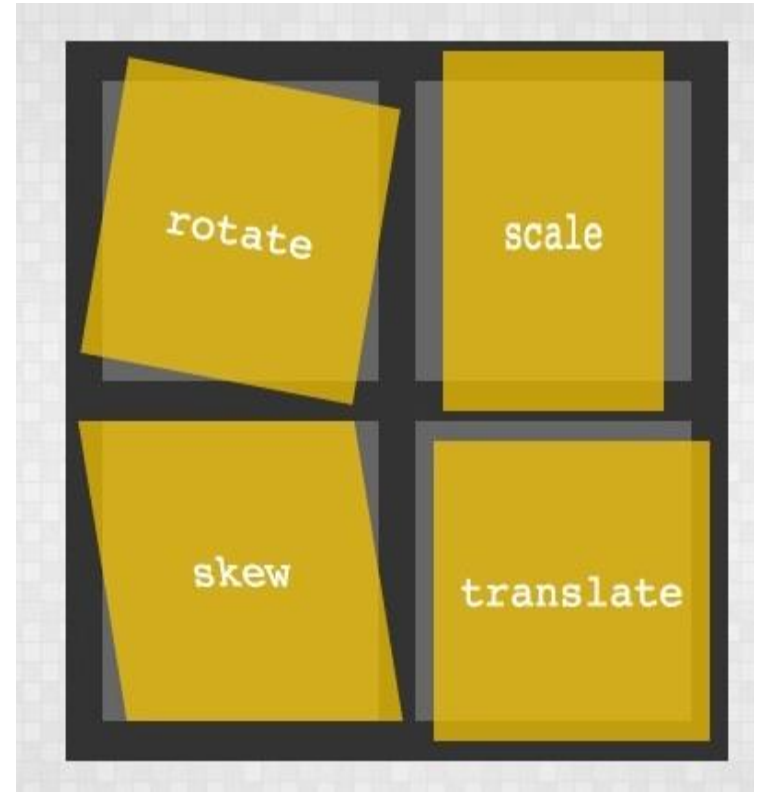
Transformation is powerful way to manipulate the shape, size, and position of a box and its contents using the transform property.

Manipulating a box over two dimensions, transform can be used to **rotate**, **skew**, **scale** and **translate** a box and its contents.

## Rotating

The following would result in a square orange box with all of its contents — text, images, whatever — obediently standing to attention:

```
.note { width: 300px; height: 300px;  
background:hsl(36,100%,50%); }
```

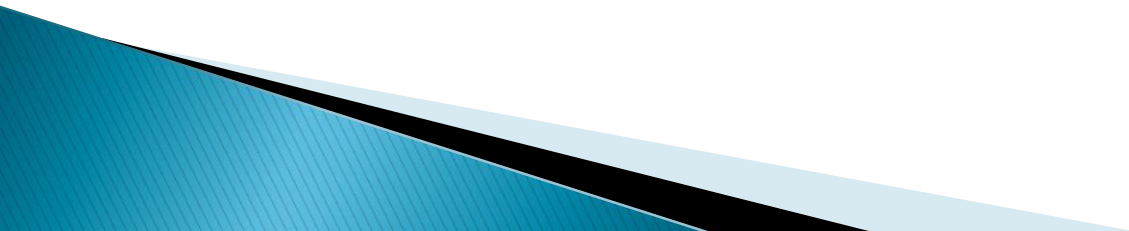


# Transform:rotate

But you can spin those soldiers around by adding a further declaration:

```
transform: rotate(-10deg);
```

This will turn the box and, importantly, everything in it, ten degrees anti-clockwise.



# Transform Skewing

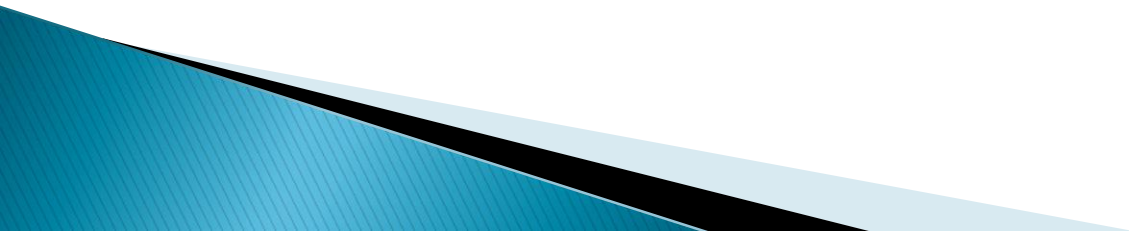
## Skewing

Skewing allows you to tip the horizontal and vertical so the following...

**transform: skew(20deg, 10deg);**

...will tip over the x-axis by 20 degrees on the y-axis by 10 degrees.

You can also specify one angle, such as skew(20deg), which is the equivalent of skew(20deg, 0).





# Transform Scaling

## Scaling

You can change width and height properties on a box, but that won't affect the size of anything inside it. Scaling, however, will multiply not only the width and height, but the size of everything contained in the box as well:

```
transform: scale(2);
```

This will multiply the size by two. You can use any positive number, including a value less than “1”, such as “0.5”, if you want to use a shrink ray.

You can also scale the horizontal and vertical dimensions separately:

```
transform: scale(1,2);
```



# Transform :Translating

## Translating

You can shift a box horizontally and vertically using `transform: translate`:

```
transform: translate(100px,200px);
```

Similar to `position: relative; left: 100px; top: 200px;`, this will move a box 100 pixels to the right and 200 pixels down.

## Combining transformations

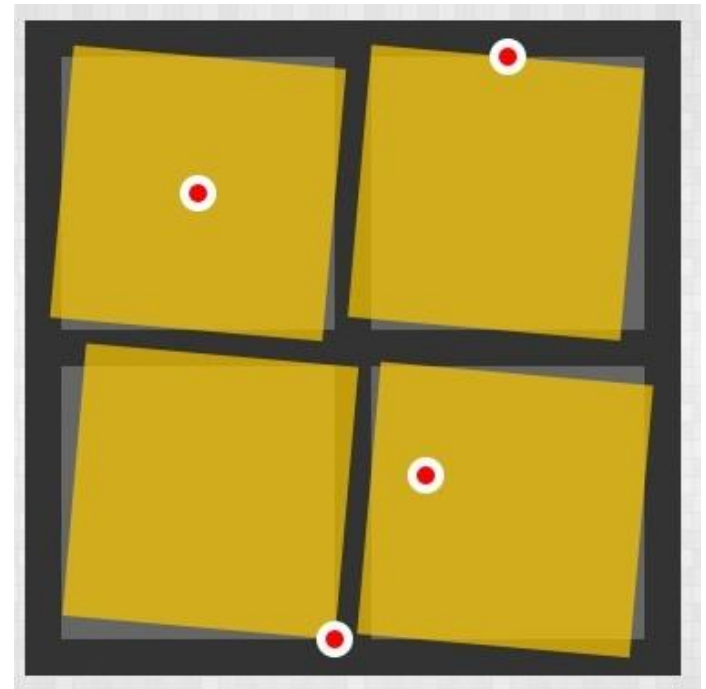
Want to rotate and scale at the same time? You crazy kid. You can do this by simply separating values with spaces, such as:

```
transform: rotate(-10deg) scale(2);
```



# Transform :Origin

- ▶ If you are transforming a box, there is a further parameter involved: the **origin of the transformation**. If you are rotating, for example, it will, by default, turn on the point that is the center of the box. You can change where in the card the pin is stuck with `transform-origin`, however:
- ▶ `transform-origin: 0 0;`



# Gradient

- ▶ There is no special property for this; you simply use the **background** or **background-image** property and define your gradient in its value. You can create both **linear** and **radial** gradients this way.

## Linear gradients

- ▶ For an even spread of two colors, fading from one at the top to another at the bottom, a declaration can simply be something like:
- ▶ `background: linear-gradient(yellow, red);`



# Linear Gradient

To manipulate the angle of the fading, you slot in “to” and the destination you want the transition to head to. You can head to one side:

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

Or one corner:

```
background: linear-gradient(to bottom right, orange, red);
```

Or any angle that tickles your fancy:

```
background: linear-gradient(20deg, orange, red)
```

- ▶ And why stop at two colors? Specify as many as you dare:
- ▶ 

```
background: linear-gradient(hsl(0,100%,50%),hsl(60,100%,50%),hsl(120,100%,50%),hsl(180,100%,50%),hsl(240,100%,50%),hsl(300,100%,50%));
```

# Radial gradient

Radial gradients, with one color starting from a central point and fading to another color, use a similar syntax:

**background:**

**radial-gradient(yellow, green);**

- ▶ You can also specify the shape of the fade. By default it is an ellipse, spreading to fill the background box, but you can force it to be circular, regardless of the shape of the box:

**background: radial-gradient(circle, yellow, green);**



# CSS Animation

- ▶ CSS animations allows animation of most HTML elements without using JavaScript or Flash!

## What are CSS Animations?

- ❑ An animation lets an element gradually change from one style to another.
- ❑ You can change as many CSS properties you want, as many times you want.
- ❑ To use CSS animation, you must first specify some keyframes for the animation.
- ❑ Keyframes hold what styles the element will have at certain times.

## The @keyframes Rule

- ❑ When you specify CSS styles inside the @keyframes rule, the animation will gradually change from the current style to the new style at certain times.
- ❑ To get an animation to work, you must bind the animation to an element.

# CSS Animation

```
div {  
  width: 100px;  
  height: 100px;  
  background-color: red;  
  animation-name: example;  
  animation-duration: 4s;  
}
```

```
@keyframes example {  
  from {background-color: red;}  
  to {background-color: yellow;}  
}
```

```
</style>
```

```
</head>
```

```
<body>
```

```
<p><b>Note:</b> This example does not work in Internet Explorer 9 and earlier  
versions.</p>
```

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

```
<p><b>Note:</b> When an animation is finished, it changes back to its original  
style.</p>
```

```
</body>
```

```
</html>
```



# CSS Animation

- ▶ **Note:** The `animation-duration` property defines how long time an animation should take to complete. If the `animation-duration` property is **not specified**, no animation will occur, because the default value is 0s (0 seconds).
- ▶ In the example above we have specified when the style will change by using the keywords "from" and "to" (which represents 0% (start) and 100% (complete)).
- ▶ It is also possible to use percent. By using percent, you can add as many style changes as you like.

## Delay an Animation

- ▶ The `animation-delay` property specifies a delay for the start of an animation.
- ▶ 

```
{  
  animation-name: example  
  animation-duration: 4s;  
  animation-delay: 2s;  
}
```

# CSS Animation

## Set How Many Times an Animation Should Run

- ▶ The `animation-iteration-count` property specifies the number of times an animation should run.
- ▶ The following example will run the animation 3 times before it stops:

```
}  
animation-name: example;  
animation-duration: 4s;  
animation-iteration-count: 3;  
}
```

# CSS Animation

The **animation-direction** property specifies whether an animation should be played forwards, backwards or in alternate cycles.

The **animation-direction** property can have the following values:

**normal** – The animation is played as normal (forwards).  
This is default

**reverse** – The animation is played in reverse direction (backwards)

**alternate** – The animation is played forwards first, then backwards

**alternate-reverse** – The animation is played backwards first, then forwards

```
}  
animation-name: example  
animation-duration: 4s;  
animation-direction: reverse;  
}
```

# CSS Animation

The **animation-timing-function** property specifies the speed curve of the animation.

The **animation-timing-function** property can have the following values:

**ease** – Specifies an animation with a slow start, then fast, then end slowly (this is default)

**linear** – Specifies an animation with the same speed from start to end

**ease-in** – Specifies an animation with a slow start

**ease-out** – Specifies an animation with a slow end

**ease-in-out** – Specifies an animation with a slow start and end

**cubic-bezier(n,n,n,n)** – Lets you define your own values in a cubic-bezier function