What You’ll Learn

- The `word-wrap` property to contain overflowing text
- The `border-radius` property to create rounded corners
- HSLA to create semitransparent backgrounds
- The `linear-gradient` function to create gradient backgrounds
- The `box-shadow` property to create drop shadows behind objects
- The `text-shadow` property to create drop shadows behind text
- The `transform` property to rotate objects
The Base Page

- It’s important to make sure your pages are functional and at least decent-looking in browsers that don’t support CSS3 before you add on CSS3.

- So before delving into any CSS3 fanciness, you’d want to get some basic styles in place to take care of older, non-CSS3-supporting browsers.
The Base Page

- We’re going to look at styling a blog’s comments’ section. We’ll start with some basic styles applied.

- The text, avatar image, commenter’s name, and date for each comment have been laid out neatly, the text is formatted, and we even have some basic backgrounds and borders in place.

- There’s nothing wrong with this comments area; it’s usable, it’s clean, it’s attractive. Anyone seeing it in an older browser would not think they were missing something or that the page was “broken.”
Problem: It’s not uncommon for people to include URLs in comments and posts, and these URLs often overflow their containers due to their length.

If the URLs have dashes (-) in them, all the major browsers can wrap the text of the URLs just fine. But Webkit-based browsers and IE will not wrap at the forward-slash (/) character, and none of the major browsers will wrap at underscores (_).
Corralling Long Text

- In CSS3, there’s finally an easy way to tell the browser to wrap text within words and stop it from overflowing.

- All you have to do is give the `word-wrap` property a value of `break-word`, and the browser will wrap text within a word if it has to in order to keep it from overflowing.
The **word-wrap** Property

- The word-wrap property is part of the Text module found at [www.w3.org/TR/css3-text](http://www.w3.org/TR/css3-text).

- It controls whether or not text is allowed to break within “words.” (The separate text-wrap property controls how lines break between words.)

- The word-wrap property can be set either to `normal` (the default) or `break-word`. 
The **word-wrap** Property

- Other than breaking long URLs, you might want to use word-wrap for keeping data tables from becoming too wide and overflowing or breaking your layout; see [www.456bereastreet.com/archive/200704/how_to_prevent_html_tables_from_becoming_too_wide](http://www.456bereastreet.com/archive/200704/how_to_prevent_html_tables_from_becoming_too_wide)
The **word-wrap** Property

- In graphics.css, add the word-wrap property to the blockquote rule:

```css
blockquote {
  margin: 0 0 0 112px;
  padding: 10px 15px 5px 15px;
  border-top: 1px solid #fff;
  background-color: #A6DADC;
  word-wrap: break-word;
}
```

- The browser will still try to wrap first at normal breakpoints, but if it has to, it will now wrap the text at underscores or even within a word.
Rounding the Corners

- Rounded corners are a simple, common visual effect that used to be surprisingly hard to create in an actual web page.

- Creating the rounded-corner images in a graphics program was time-consuming, as was creating the HTML and CSS.

- You’d often have to add extra nested divs to place each corner image separately, since CSS 2.1 allows only one background image per box, and the CSS used to actually control the placement of the images could get complicated.

- Even if you used a script to dynamically create the rounded corners you were still adding to the number of files that users had to download and decreasing your pages’ performance.
The **border-radius** Property

- The border-radius property is part of the Backgrounds and Borders module found at www.w3.org/TR/css3-background.

- It’s shorthand for the properties specifying the rounding amount of each of the four corners, in this order: border-top-left-radius, border-top-right-radius, border-bottom-right-radius, border-bottom-left-radius.

- You can write out all four values, with spaces in between, in one border-radius property, or just use one value to round all four corners the same amount.
The `border-radius` Property

- Other than speech bubbles, you might want to use `border-radius` for:
  - Buttons; see [http://blogfreakz.com/button/css3-button-tutorials](http://blogfreakz.com/button/css3-button-tutorials) and [http://css-tricks.com/examples/ButtonMaker](http://css-tricks.com/examples/ButtonMaker)
  - Tabs
  - Dialog boxes
  - Circular badges
  - Bar charts; see [www.marcofolio.net/css/animated_wicked_css3_3d_bar_chart.html](http://www.marcofolio.net/css/animated_wicked_css3_3d_bar_chart.html)
Rounded Corners

- In CSS3, creating rounded corners can be as simple as `border-radius: 10px` on a single div. No extra markup, no images, no JavaScript.

- We will modify the blockquote rule:

```css
blockquote {
  margin: 0 0 0 112px;
  padding: 10px 15px 5px 15px;
  -moz-border-radius: 20px;
  -webkit-border-radius: 20px;
  border-radius: 20px;
  border-top: 1px solid #fff;
  background-color: #A6DADC;
  word-wrap: break-word;
}
```
Rounded Corners

- The border-radius: 20px; declaration is the W3C standard syntax for rounded corners, specifying that all four corners should be rounded by 20 pixels.

- This syntax is supported by Opera, Chrome, Safari 5, and IE 9.

- Firefox and Safari 4 and earlier use the -moz-border-radius and -webkit-border-radius properties, respectively.
Rounded Corners

- With these three lines added, the corners are now rounded in all browsers except IE 8 and earlier.

- These versions of IE simply ignore the properties and keep the corners straight—no harm done.
Creating Ovals and Circles with border-radius (an aside)

- If you want your speech bubbles to be complete ovals instead of rounded rectangles, you’ll need to use elliptical-shaped corners instead of perfectly round ones. (Elliptical just means that the curve of each corner is somewhat flattened out—just like an oval.)

- To specify an elliptical corner, you write two measurements, separated by a slash, such as this: border-radius: 50px/20px. (Safari 3 and 4 use the non-standard syntax of no slash, just a space.)

- This means that the curve will extend horizontally 50 pixels but vertically only 20 pixels, making a flattened, elliptical curve.

- You can make each corner have different angles; find out how at http://css-tricks.com/snippets/css/rounded-corners.
Creating Ovals and Circles with border-radius (an aside)

- To create circles, first give your box the same width and height; use ems as the unit of measurement instead of pixels to ensure it can grow with its text.

- Then set each corner’s border-radius to one-half the width/height value. For instance, if you have a box that is 10 ems wide and tall, use border-radius: 5em.
Adding the Bubble’s Tail

- We can add the tail without using any graphics.

- In fact, we can add it without using any CSS3—the technique only uses properties and selectors from CSS 2.

- All we need to create a tail is a triangle, and you can create triangles with pure CSS by using regular old borders.
Adding the Bubble’s Tail

- When two borders of a box meet at a corner, the browser draws their meeting point at an angle.

- If you reduce that box’s width and height to zero, and give every border a thick width and a different colour, you’ll end up with the appearance of four triangles pushed together, each pointing in a different direction.
Adding the Bubble’s Tail

- Here’s what the CSS and HTML look like:

```css
.triangles {
    border-color: red, green, blue, orange;
    border-style: solid;
    border-width: 20px;
    width: 0;
    height: 0;
}

<div class="triangles"></div>
```
Adding the Bubble’s Tail

- What would happen if you made the top, left, and bottom borders transparent instead of coloured?

- Only the right border would show, leaving the appearance of a left-pointing triangle:

```css
.triangle-left {
    border-color: transparent green transparent transparent;
    border-style: solid;
    border-width: 20px;
    width: 0;
    height: 0;
}

<div class="triangle-left"></div>
```
Adding the Bubble’s Tail

- So, all you need to do to create a triangle using CSS is give an element zero width and height, give it thick borders, and make all but one of those borders transparent.

- You can vary the angle of the triangle by making the widths of the borders different on different sides.
Generating the Tail

- Let’s leave the HTML pristine and use CSS-generated content to make the element we need appear.

- Generated content is a CSS 2.1 technique where you place content into your CSS to have it appear in your HTML.

- It’s useful for adding things that you don’t want to manually hard-code into the HTML, like numbers before headings or icons after links.

- It shouldn’t be used for essential content that would be missed if the user couldn’t access the CSS file.
Generating the Tail

To create generated content, you need to

1. specify where the content is to be inserted, using either the ::before or ::after pseudo-elements (also written as :before and :after – CSS3 changed the syntax for pseudo-elements to use double colons, while pseudo-classes retain the single colons), and

2. specify what content to insert, using the content property.
Generating the Tail

- For instance, to insert the word “Figure” before every image caption on your page, you could use the following CSS:
  ```css
  .caption:before {
    content: "Figure: ";
  }
  ```

- This CSS would turn the HTML
  `<p class="caption">Isn’t my cat cute?</p>` into this text when seen on the page:
  ```
  Figure: Isn't my cat cute?
  ```
Generating the Tail

- In the case of the speech-bubble tail we want to generate, all we want to see are the borders of the generated content, not the content itself. So, let’s generate a piece of invisible content: a non-breaking space.

- The HTML entity for a non-breaking space is \&nbsp\; but you can’t use HTML entities within the content property. Instead, you need to use the hexadecimal part of the character’s Unicode code point (or reference).

- There are lots of handy charts online that allow you to look up this kind of stuff.
Generating the Tail

- E.g., at [www.digitalmediamminute.com/reference/entity](http://www.digitalmediamminute.com/reference/entity) you can see 252 little boxes, each showing one of the allowed entities in (X)HTML.

- In the “Filter entities by keyword” box, type “non-breaking” 251 of the boxes will disappear, leaving you with one box showing `&nbsp;`, the HTML entity name.

- Position your cursor over the box. Two other codes will appear: its numerical code (in this case, `&#160;`) and its Unicode code (`u00A0`).

- You just want the hexadecimal part of the Unicode code, which is the part after the “u.” Copy the text “00A0”.
Generating the Tail

- Even though we now have the Unicode code we need, we can’t put it straight into the content property, like so:

```css
blockquote:after {
  content:”00A0”;
}
```

- If we did this, the browser would quite logically make the text “00A0” show up, instead of the non-breaking space.
Generating the Tail

- To tell the browser that we’re putting in a special character code, we need to escape the code. This means you have to put a backslash in front of the code.

- This alerts the browser that what follows the slash is not to be taken as literal text, but is instead a code for something else.

```css
blockquote:after {
  content: "\00A0";
}
```
Generating the Tail

- Now we need to add borders around the non-breaking space to make it show up.

- We also need to set its width and height to zero and make it display as a block element so we can move it around to place the tail against the side of the speech bubble:

```css
blockquote:after {
  content: "\00a0";
  display: block;
  width: 0;
  height: 0;
  border-width: 10px 20px 10px 0;
  border-style: solid;
  border-color: transparent #000 transparent transparent;
}
```
Generating the Tail

- Now we will move it with absolute positioning. First, add `position: relative;` to the blockquote rule; this establishes it as the reference point for the absolute element’s positioning:

```css
blockquote {
    position: relative;
    margin: 0 0 0 112px;
    padding: 10px 15px 5px 15px;
    -moz-border-radius: 20px;
    -webkit-border-radius: 20px;
    border-radius: 20px;
    border-top: 1px solid #fff;
    background-color: #A6DADC;
    word-wrap: break-word;
}
```
Generating the Tail

- Then, add the absolute positioning to the generated content, along with top and left values:

```css
blockquote:after {
  content: "\00a0";
  display: block;
  position: absolute;
  top: 20px;
  left: -20px;
  width: 0;
  height: 0;
  border-width: 10px 20px 10px 0;
  border-style: solid;
  border-color: transparent #000 transparent transparent;
}
```
Generating the Tail

- It’s possible that a comment might be so short that the tail hangs off the bottom, as seen in the second comment.

- To fix this, add `min-height: 42px;` to the blockquote rule.

- Now that the triangle isn’t layered over the blockquote, we can change its colour to match the blockquote:

  ```css
  border-color: transparent #A6DADC
  transparent transparent transparent;
  ```
Semitransparent Backgrounds with RGBA or HSLA

- One great way to add depth is to make backgrounds semitransparent (also called alpha transparency).

- By letting a little bit of the page background show through, you create more of a layered appearance, as if the semitransparent element is floating over the background.

- Before CSS3, you could create semitransparent backgrounds using an alpha-transparent PNG as a tiling background image.
CSS3’s RGBA and HSLA Syntax

- Luckily, in CSS3 we have both RGBA and HSLA to turn to.

- Both are methods for specifying a colour and its level of transparency at the same time.

- RGBA stands for red-green-blue-alpha (for alpha transparency)

- HSLA stands for hue-saturation-lightness-alpha.
We could specify the shade of blue that we’re using as the speech bubble’s background using any of these syntaxes:

- Hexadecimal: #A6DADC
- RGB: 166, 218, 220
- RGBA: 166, 218, 220, 1
- HSL: 182, 44%, 76%
- HSLA: 182, 44%, 76%, 1
CSS3’s RGBA and HSLA Syntax

- In the RGBA syntax, the first three values are the amounts of red, green, and blue, either from 0%–100% or, more commonly, 0–255.

- In the HSLA syntax, the first three values are the hue value, from 0 to 360; the percentage level of saturation; and the percentage level of lightness.

- In both RGBA and HSLA, the fourth value is the opacity level, from 0 (completely transparent) to 1 (completely opaque).
The colour converter tool at http://serennu.com/colour/hsltorgb.php allows you to convert colour values you already have into hex, RGB, and HSL syntaxes.

The Doughnut Color Picker at http://www.workwithcolor.com/hsl-color-picker-01.htm lets you both pick and convert colours. The picker uses HSL, but gives the hex and RGB equivalents, and lets you input colours in any of the three syntaxes.
Creating Semi-transparent Speech Bubbles

- We need to switch the speech bubbles’ background colour from hexadecimal to HSLA notation and make them semitransparent.

- The speech bubbles’ background colour is currently set to #A6DADC.

- The HSLA equivalent is hsl(182, 44%, 76%).

```css
background-color: hsl(182, 44%, 76%);
```
Creating Semi-transparent Speech Bubbles

- Now we’ll modify this new syntax to make the speech bubbles semitransparent.

- Change `background-color: hsl(182, 44%, 76%);` to `background-color: hsla(182, 44%, 76%, .5);`

- To change the tail to match, copy and paste the HSLA value over the hexadecimal value in the `border-color` declaration:
Image-free Gradients

- CSS3 allows you to create gradients without images, speeding up your development time and decreasing page-loading times.

- CSS-generated gradients also have the advantage of being able to scale with their containers in ways that image gradients can’t, making them more versatile.

- You can create both linear (straight) gradients and radial (circular or elliptical) gradients.
There is no gradient property; you specify a gradient using the linear-gradient or radial-gradient function as the value for any property that allows an image value, such as background-image and list-style image.

When you specify a linear gradient, you tell the browser its starting point, angle, and start and end colours.

You can also add extra colours in between the start and end colours and specify the exact position of each colour along the line of the gradient.
Unfortunately, Firefox and Webkit differ on the syntax required to feed the browser this information; Firefox matches the official W3C syntax, and Webkit uses a very different (and more complicated) syntax that they developed first.

Not only that, but even within each single syntax there are many variations on how you can specify the same gradient. It can get pretty confusing.

https://www.w3schools.com/css/css3_gradients.asp
#grad {
    background: red; /* For browsers that do not support gradients */
    background: -webkit-linear-gradient(red, yellow); /* For Safari 5.1 to 6.0 */
    background: -o-linear-gradient(red, yellow); /* For Opera 11.1 to 12.0 */
    background: -moz-linear-gradient(red, yellow); /* For Firefox 3.6 to 15 */
    background: linear-gradient(red, yellow); /* Standard syntax */
}
The Firefox & W3C Syntax

- Firefox’s syntax matches the official syntax being developed by the W3C and is generally easier to understand and use.

- First, add a linear gradient for Firefox in the background-image property of the blockquote rule, using the `-moz-linear-gradient` function:

```css
background-image: -moz-linear-gradient(
    hsla(0,0%,100%,.6),
    hsla(0,0%,100%,0) 30px
);
```
The Firefox & W3C Syntax

- This specifies a start colour \((hsla(0,0\%,100\%,.6))\), end colour \((hsla(0,0\%,100\%,0))\), and the position of the end colour \((30px)\).

- Because we haven’t specified any starting point for the gradient or its angle, Firefox will simply use the default values, which makes the gradient start at the top of the box and run straight down.

- The start colour is white at 60 percent opacity, and the end colour is white at zero percent opacity (completely transparent). Laying semitransparent white over the background colour creates a tint of whatever that background colour is.
For Webkit-based browsers, add another background-image declaration to the blockquote rule, this time containing the `-webkit-gradient` function:

```css
background-image: -webkit-gradient(linear,
    0 0, 0 30,
    from(hsla(0,0%,100%,.6)),
    to(hsla(0,0%,100%,0))
);
The Webkit Syntax

- First, you specify the type of gradient - linear or radial - within the -webkit-gradient function itself.

- Next, you specify the horizontal and vertical positions of the start point (here, 0 0), followed by the horizontal and vertical positions of the end point (here, 0 30).

- So, we’re telling Webkit that we want the gradient to start at a point zero pixels across and zero pixels down the box (the top left corner) and end at zero pixels across and 30 pixels down the box.

- After the start and end points, we have the starting colour and the end colour, but note that you must include from and to before each colour.
Image-free Gradients

- The CSS syntax differences between Firefox and Webkit can be hard to remember.

- Luckily, there are some great gradient-generator tools online that allow you to use a visual editor to create the gradient, and then they write the corresponding CSS you need to use. Just copy and paste!

- Find these gradient generators at
  - http://westciv.com/tools/gradients
  - http://westciv.com/tools/radialgradients,
  - www.display-inline.fr/projects/css-gradient
Image-free Drop Shadows

- Drop shadows on boxes are created in CSS3 using the box-shadow property.

- In the property, you set the shadow’s horizontal and vertical offsets from the box, its colour, and you can optionally set blur radius as well as spread radius.
Image-free Drop Shadows

- Add the following three lines of CSS to the blockquote rule:

  - `moz-box-shadow: 1px 1px 2px hsla(0,0%,0%,.3);`
  - `webkit-box-shadow: 1px 1px 2px hsla(0,0%,0%,.3);`
  - `box-shadow: 1px 1px 2px hsla(0,0%,0%,.3);`

- Just as with border-radius, all three lines accomplish the same thing, but are read by different browsers
Image-free Drop Shadows

- The first value in each property, 1px, is the horizontal offset from the box, and it tells the browser to move the shadow one pixel to the right of the box’s edge.

- The second value, 1px, is the vertical offset, moving the shadow one pixel down.

- You can use negative values to move the shadow to the left and up instead.
Image-free Drop Shadows

- The third value, 2px, is the blur radius, which specifies over how many pixels the shadow should stretch.

- A larger value makes the shadow blurrier and softer; a value of zero would make it completely sharp-edged.

- The fourth value is the colour—in this case, black at 30 percent opacity.
Image-free Drop Shadows

- You can use any syntax for declaring the colour in box-shadow, but HSLA or RGBA - the only syntaxes that can make a colour semitransparent - are your best bets.

- Semitransparency is very handy for drop shadows, since you want to be able to see the background of whatever is behind the shadow peeking through a bit.
3D Hover Effect

- We can increase the 3D appearance by making the speech bubbles appear to move forward a bit when each is hovered over.

- The farther away the speech bubble is from the background, the larger its shadow should appear.
3D Hover Effect

- You increase the offset of the shadow on hover by adding this rule:

```css
blockquote:hover {
  top: -2px;
  left: -2px;
  -moz-box-shadow: 3px 3px 2px hsla(0,0%,0%,.3);
  -webkit-box-shadow: 3px 3px 2px hsla(0,0%,0%,.3);
  box-shadow: 3px 3px 2px hsla(0,0%,0%,.3);
}
```
3D Hover Effect

- The negative top and left values are what actually shift the speech bubble and create the appearance of movement, but increasing the shadow as well—from 1 pixel offset to 3 pixels offset—makes the movement look more realistic.

- Increasing the shadow also makes it appear more like the speech bubble is moving away from the background and closer to the user, instead of just farther up the page.
The box-shadow Property

- The box-shadow property is part of a draft of the Backgrounds and Borders module; this draft is found at http://dev.w3.org/csswg/css3-background/#the-box-shadow, but ultimately the finalized module can be found at www.w3.org/TR/css3-background.
The box-shadow Property

- You can make a shadow appear inside a box instead of outside or behind it using the inset keyword, added at the start or end of the box-shadow value.

- Spread radius and inset are not supported in Safari 4 and earlier, Safari on iOS 3 and earlier, or IE 9.

- You can apply multiple shadows to the same box by writing each in the same box-shadow property, separated by commas. They’ll be stacked on top of each other, with the first shadow declared on top.
The box-shadow Property

- Other than creating basic shadows behind boxes, you might want to use box-shadow for:
  - Glows (by not offsetting the shadow at all and also optionally using a positive spread radius value)
  - 3D-looking buttons
  - Simulating multiple borders around a box (using multiple box-shadows, each set to 0 blur radius to give them hard edges); see http://weston.ruter.net/2009/06/15/multiple-borders-via-css-box-shadow
  - Simulating gradients (using inset box-shadow); see http://nimbupani.com/vignettes-with-css3-box-shadows.html
Let’s add a text-shadow on hover to highlight the chosen comment just a bit by adding the following line to the blockquote:hover rule:

text-shadow: 1px 1px 1px hsla(0,0%,100%,.7);

The syntax is almost exactly the same as the syntax for box-shadow. (The only difference is that you can’t set spread radius or inset on text-shadow.)

We have a horizontal offset, vertical offset, optional blur radius, and colour.

In this case, there’s no need to add any browser-specific prefixes; Firefox, Safari, Chrome, and Opera all support the standard text-shadow property.
Image-free Text Shadows

- Another nice place to add a shadow behind text is the commenter’s name and date of comment.

- These two pieces of text are pretty small and are sitting on top of a patterned background. A very slight, sharp edged text shadow would give it a subtle outline to make it stand out more and be a little easier to read.

- Add the following line to the existing .comment-meta rule:
  
  `text-shadow: 1px 1px 0 hsla(0,0%,100%,.7);`
The `text-shadow` Property

- The `text-shadow` property is part of the Text module found at [www.w3.org/TR/css3-text](http://www.w3.org/TR/css3-text). It was part of CSS 2, removed from 2.1, and is back in 3.

- In the property, you set the shadow’s colour and its horizontal and vertical offsets from the text.

- You can also set a blur radius; the default (if you leave it out) is zero.

- You can apply multiple shadows to the same text by writing each in the same `text-shadow` property, separated by commas. They’ll be stacked on top of each other, with the first declared shadow on the top.
The text-shadow Property

- Other than creating basic shadows behind text, you might want to use text-shadow for:
  - Glows;
  - Letterpress, engraved, cut-out or embossed text (using a light shadow on one side of the text and a dark shadow on the other side); see [http://sixrevisions.com/css/how-to-create-inset-typography-with-css3](http://sixrevisions.com/css/how-to-create-inset-typography-with-css3)
  - Fiery text (using multiple yellow, orange, and red shadows); see [www.css3.info/preview/text-shadow](http://www.css3.info/preview/text-shadow)
  - Blurred text (using a shadow the same colour as the text, or simply transparent colour for the text);
  - Creating the appearance that text is stacked into a 3D column (using multiple shadows); see [http://css-tricks.com/3d-text-tower](http://css-tricks.com/3d-text-tower)
  - Creating the appearance that links are pushed inwards like a button when clicked, by decreasing text-shadow; see [www.impressivewebs.com/text-shadow-links](http://www.impressivewebs.com/text-shadow-links)

- Also check out [http://maettig.com/code/css/text-shadow.html](http://maettig.com/code/css/text-shadow.html) for many examples of text-shadow effects; some are more practical than others.
Transforming the Avatars

- **Transforms** are a collection of effects, each called a transform function, that manipulate the box in ways like rotating, scaling, and skewing.

- These effects would previously have had to be accomplished with images, Flash, or JavaScript.

- Transforming objects with pure CSS avoids the need for these extra files, once again increasing the efficiency of both your development and the pages themselves.
The **transform** Property

- The **transform** property is part of both the 3D Transforms module found at [www.w3.org/TR/css3-3d-transforms](http://www.w3.org/TR/css3-3d-transforms), and the 2D Transforms module, at [www.w3.org/TR/css3-2d-transforms](http://www.w3.org/TR/css3-2d-transforms).

- All of the 2D transform functions are also included in the 3D spec, so you may just want to refer to the 3D spec.

- There are too many transform functions to list here, but here’s sample syntax for the most important and supported ones:
The **transform** Property

- **translate**
  - moves the object to a new location, specified as an X and Y coordinate.
  - Positive values move it right and down, respectively, and negative values move it left and up.
  - Example: `translate(20px, -10px)`
The \textbf{transform} Property

- \textbf{scale}
  - Scales the dimensions of the object \textit{X} number of times.
  - Negative values flip the object.
  - To scale to something smaller, use a number between 0 and 1.
  - If you use two values, separated by commas, the first is the horizontal scaling factor and the second is the vertical scaling factor.
  - Example: \texttt{scale(2.5)} or \texttt{scale (1, .5)}
The **transform** Property

- **rotate**
  - Turns an object a specified number of degrees (deg).
  - Positive values turn it clockwise; negative values turn it counter clockwise.
  - Example: `rotate(45deg)`

- **skew**
  - Skews or warps an object, again in degrees.
  - The first value controls the horizontal slope and the second the vertical.
  - If you use only one value, the skew on the Y axis is set to zero.
  - Example: `skew(10deg, 20deg)`
The **transform** Property

- You can include multiple transform functions in one transform property, separated with spaces. The transforms are applied in the order listed.

- You can use the transform-origin property to specify the point of origin from which the transform takes place, using keywords, numbers, or percentages. The default is the centre.

- When you transform an object, the other objects around it don’t move to make way for the transformation (similar to relative positioning). The object is placed first in the flow, and then transformed.
The **transform** Property

- Other than rotating avatars, you might want to use transforms for:
  - Increased link, button, or table row size on hover
  - Display of an image gallery where thumbnails scale up when hovered
  - Angled photos (to create the appearance of that they’ve been tacked-up or are scattered across a table, for instance)
  - Angled sticky-note-style boxes
  - Randomly angled tags in a tag cloud; see [http://code.almeros.com/how-to-create-a-css3-enabled-tag-cloud](http://code.almeros.com/how-to-create-a-css3-enabled-tag-cloud)
  - Skewed boxes or images (to imply perspective)
The **transform** Property

- Small diagonal banner in the top corner of a page
- Sideways text (popular in date stamps on blog posts, for instance); see [http://snook.ca/archives/html_and_css/css-text-rotation](http://snook.ca/archives/html_and_css/css-text-rotation)
- Printable folding card; see [http://natbat.net/2009/May/21/pocketbooks](http://natbat.net/2009/May/21/pocketbooks)
- Slideshow where images slide in and out of viewing window (using translate); see [http://css3.bradshawenterprises.com/#slide2](http://css3.bradshawenterprises.com/#slide2)
- Links or tabs that slide up into full view when hovered (using translate); see [http://creativefan.com/css3-tutorial-create-cardpockets-how-to](http://creativefan.com/css3-tutorial-create-cardpockets-how-to)
Let’s look at the syntax for transforms by rotating the avatars. Add this new rule to the styles:

```css
.comment-meta img {
  -moz-transform: rotate(-5deg);
  -o-transform: rotate(-5deg);
  -webkit-transform: rotate(-5deg);
  transform: rotate(-5deg);
}
```
Rotating the Avatars

- The transform property (and, for now, all three browser-specific equivalents) tells the browser that you want to apply a transform.

- You then specify that the particular transform function you want is rotate, and that the number of degrees of rotation, using the deg unit, is negative five. You can use either positive or negative values.

- Other transform functions take different types of measurements but the pattern is always the same:

  ```javascript
  transform: function(measurements);
  ```