CS7026 – CSS3

CSS3 – Attribute Selectors
What are Attribute Selectors?

- Attribute selectors are powerful and useful because they allow you to target specific elements without needing IDs or classes in the HTML.

- Instead, attribute selectors target an element based on the existence or value of a specific attribute on that element.
What are Attribute Selectors?

- For instance, the selector `img[alt]` is made up of the type selector `img` followed by the attribute selector `[alt]`.

- All attribute selectors are designated by square brackets, but what goes in the brackets depends on what you’re trying to target.

- The `img[alt]` selector targets all `img` elements that have an `alt` attribute present.

- Using this selector while testing your pages, you could give all images that have `alt` attributes a bright green outline, so you could see at a glance which images don’t have the outline and need `alt` attributes added.
What are Attribute Selectors?

```css
img[alt] {  
border: 3px solid #0C0;
}

<img src="images/cat.jpg" width="320" height="241" alt="My cat Poe">
<img src="photos/dog.jpg" width="320" height="240">
What are Attribute Selectors?

- The `img[alt]` selector is an example of the simplest type of attribute selector - one that checks only for the presence of an attribute, regardless of its value.

- It’s one of the four types of attribute selectors that are in the CSS 2.1 spec.
CSS 2.1 Attribute Selectors

<table>
<thead>
<tr>
<th>ATTRIBUTE SELECTOR</th>
<th>FUNCTION</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>[attr]</code></td>
<td>Matches an element with an <code>attr</code> attribute present, regardless of its value.</td>
<td></td>
</tr>
<tr>
<td><code>[attr=val]</code></td>
<td>Matches an element with an <code>attr</code> attribute whose value is exactly <code>val</code>.</td>
<td></td>
</tr>
<tr>
<td><code>[attr~=val]</code></td>
<td>Matches an element with an <code>attr</code> attribute whose value is a space-separated list of words, one of which is exactly <code>val</code>.</td>
<td></td>
</tr>
<tr>
<td>`[attr</td>
<td>=val]`</td>
<td>Matches an element with an <code>attr</code> attribute whose value is either exactly <code>val</code> or begins with <code>val</code> immediately followed by a hyphen.</td>
</tr>
</tbody>
</table>

- The W3C calls these CSS 2.1 attribute selectors *attribute presence and value selectors*.  
CSS3 Attribute Selectors

- CSS3 introduces three new attribute selectors that offer even more fine-grained control over what you’re trying to target.

<table>
<thead>
<tr>
<th>ATTRIBUTE SELECTOR</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>[attr^=val]</td>
<td>Matches an element with an attr attribute whose value starts with val.</td>
</tr>
<tr>
<td>[attr$=val]</td>
<td>Matches an element with an attr attribute whose value ends with val.</td>
</tr>
<tr>
<td>[attr*=val]</td>
<td>Matches an element with an attr attribute whose value contains val somewhere within it.</td>
</tr>
</tbody>
</table>

- The W3C calls these CSS3 attribute selectors *substring matching attribute selectors* because they match a part of a value instead of the whole thing.
The Base Page

- The page that we’ll be using as our starting point is the finished page from a previous lecture.

- It contains a lot of links to different types of files, but all of these links are styled the same right now.

- It would be nice if links to certain file types were styled differently, to give the user a visual cue to the type of document they’re about to open.

- The page also contains several images, most of which are photos, but one of which is a thumbnail of a calendar. Again, it would be nice to style photos differently than other images, but right now all these images are styled the same way.
Indicating File Types with Dynamically Added Icons

- Open the hpage. Throughout the page, there are links to documents to download, in these file types:
  - PDF
  - MOV
  - DOC
  - JPG

- We are going to add some sort of file-type indicator by using attribute selectors.

- Every link ends with a file-type extension, so we can use the "end of the value" attribute selector to examine the extension and add the appropriate icon as a background image on the a element.
First, prepare the `a` elements inside the file-download lists to have background images added to them:

```html
ul a {
  display: block;
  min-height: 15px;
  padding-left: 20px;
  background-repeat: no-repeat;
  background-position: 0 3px;
}
```
Indicating File Types with Dynamically Added Icons

- This makes the links block elements with a minimum height matching the height of the icon images, so the icons won’t ever get cut off.

- It also adds left padding to create empty space for each icon to sit in.

- Each icon background image will display only once (no-repeat) and be positioned three pixels down from the top of the link (0 3px) to align it with the top of the text.
Now we can add the attribute selectors to target each file type extension:

```html
a[href$=".pdf"] {
  background-image: url(images/icon_pdf.png);
}
a[href$=".doc"] {
  background-image: url(images/icon_doc.png);
}
a[href$=".mov"] {
  background-image: url(images/icon_film.png);
}
a[href$=".jpg"] {
  background-image: url(images/icon_photo.png);
}
```
Indicating File Types with Dynamically Added Icons

- The `href$=` part of each attribute selector tells the browser "find every `href` attribute that ends with," and then the values in quotation marks, such as `.pdf`, give the ending attribute value to match against.

- When the browser finds a match, it applies the background image indicated, adding appropriate icons to all the links.
Alternative Icon Ideas

- The icons are a nice little hint to help your users, but if you wanted to be even more obvious and explicit, you could use generated content to write out the file-type extension at the end of each link instead of or in addition to the icons.

- You’d first need to make sure that this information wasn’t already manually written in each link.
Then, you could add the following rule, for example, to write out "(PDF)" after each link to a PDF file:

```css
a[href$=".pdf"]:after {
  content: " (PDF)";
}
```
Combining Multiple Attribute Selectors

- As with almost any other type of selector, you can combine multiple attribute selectors into one to give you even more fine-grained control over what you want to target.

- For instance, what if you wanted to show the photo icon for links to PNG images, but a chart icon for links to PNG images that also happened to be charts?

- Depending on how your images are named, this selector would work:

  ```css
  a[href$=".png"][href*="chart"] {
    background-image: url(images/icon_chart.png);
  }
  ```
Combining Multiple Attribute Selectors

- This selector tells the browser "find all links that have ‘.png’ at the end of their **href** attributes and have ‘chart’ somewhere in the **href** attribute."

- So all of the following links would get matched:
  - `<a href="images/chart_locations.png">`
  - `<a href="images/piechart.png">`
  - `<a href="charts/travel.png">`
Styling Full-size Photos and Thumbnails Differently

- Another great use of attribute selectors in our page is to give the photos a different style than the calendar thumbnail.

- To do this without CSS3, we could simply give the thumbnail a class and apply unique styles to this class.

- This would be quite easy in this particular page. But using classes is not always so simple in the real world.
The Trouble with Classes

- While classes have many legitimate uses, they do have some problems that make them difficult to use in some situations.

  - **Classes add bulk to your HTML.**
    - In our example, adding one class isn’t going to hurt anyone, but in much larger pages and sites with more complex styles, a lot of extra classes could be necessary, adding a good chunk to the file size.
    - Any time you can avoid adding classes and IDs to the HTML and use another way to reliably target elements instead, you should do so.
The Trouble with Classes

- **Markup may be controlled by a CMS or plugin,** making it impossible for you to add classes to the HTML.

- **Your client may be the one adding content,** and you can’t count on them to remember to assign the proper classes.

- **You may not be allowed to touch the HTML** if you’re just the CSS developer on a project, or if you’ve been brought into an existing project just to make a few style updates.
The Trouble with Classes

- **Classes can be time-consuming** to add to an existing site with tons of pages, if you’re trying to go back and add new styles. It’s much easier to write CSS that takes advantage of whatever HTML is already there, without your having to go back and add extra style hooks into the HTML.
As long as there is some reliable difference between the HTML used for the thumbnails and the photos, we can tap into that difference with attribute selectors.

In this case, the distinction is that the calendar thumbnail is saved in the folder named "thumbnails" and the photos are saved in the folder named "photos."

The folder name is part of the path in the `src` attribute, so we can use attribute selectors to target each image type independently via particular `src` attribute values.
Let’s start by floating the calendar thumbnail left instead of right:

```css
img[src*=thumbnails] {
  float: left;
  margin: 0 20px 10px 0;
}
```

The * attribute selector tells the browser "find every src attribute that has ‘thumbnails’ somewhere within it." This matches the calendar image:

```html
<img src="images/thumbnails/calendar.jpg" width="90" height="90" alt=""/>
```
Now let’s add some styling to the photos to make them look like Polaroid pictures. Add the following new rule:

```css
img[src*=photos] {
  padding: 5px 5px 30px 5px;
  background: #fff;
  -moz-box-shadow: 3px 6px 8px -4px #999;
  -webkit-box-shadow: 3px 6px 8px -4px #999;
  box-shadow: 3px 6px 8px -4px #999;
  -moz-transform: rotate(2deg);
  -o-transform: rotate(2deg);
  -webkit-transform: rotate(2deg);
  transform: rotate(2deg);
}
```
Using Attribute Selectors to Target by Type

- Now all the photos have a white border around them, a drop shadow behind them, and a slight angle.
More about Attribute Selectors

- Other than link icons and type-based image styling, you might want to use attribute selectors for:
  - Varying the styling of phrases in different languages (using [lang]=en, for instance).
  - Adding a visual indication to elements that have title attributes set (using [title]).
  - Removing bullets from lists within navigation divs (using div[id^=nav] to match <div id="nav-primary"> and <div id="nav-secondary">, for instance).
  - Styling links that go to external sites (using a[href^=http] or a[rel=external]), that are secure (using a[href^=https]), that go to a specific URL (such as a[href*="paypal.com"]), that open in new window (using a[target="_blank"]), or that go to your own home page (using a[href="http://myurl.com"] or a[href="/index.html"]).
More about Attribute Selectors

- Checking for empty links before launching a site; see http://fuelyourcoding.com/unconventional-css3-link-checking.
- Displaying the access key of a link (using a::after { content: ‘[‘ attr(accesskey) ‘]’ }).
- Displaying the citation source of a blockquote (using blockquote[cite]:after { content: ‘ - ‘ attr(cite) }).
- Styling blockquotes differently based on the value of their cite attributes.
- Displaying an image’s alternative text as its caption (using img[alt]:after { content: attr(alt) }).
- Creating a user style sheet to hide ads on web pages; see http://24ways.org/2005/the-attribute-selector-for-fun-and-no-ad-profit.
- Hiding rules from IE 6.
## Attribute Selectors Browser Support

<table>
<thead>
<tr>
<th></th>
<th>IE</th>
<th>FIREFOX</th>
<th>OPERA</th>
<th>SAFARI</th>
<th>CHROME</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Yes, 7+*</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

*IE 7 and later support all the attribute selectors, but are sometimes buggy. Test well.*
CS7026 – CSS3

CSS3 – Pseudo Classes & Pseudo Elements
What You’ll Learn

- The :nth-child() pseudo-class to select alternating elements
- The :nth-of-type() pseudo-class to select alternating elements of a certain type
- The :last-child pseudo-class to style the last element of a list differently
- The :target pseudo-class to style the target of a URL containing a fragment identifier
Targeting Specific Elements Without Using IDs or Classes

- Pseudo-classes and pseudo-elements can be used to select specific elements in the HTML without assigning those elements IDs or classes.

- Pseudo-classes and pseudo-elements target pieces of HTML that either don’t exist as standalone elements, or do exist but have a unique characteristic that you can’t target with the other simple selectors.
Targeting Specific Elements Without Using IDs or Classes

- E.g., you can use the :first-line pseudo-element to format the first line of a paragraph, even though that first line doesn’t have HTML tags wrapped around it.

- So, some pseudo-classes and pseudo elements are even more powerful than attribute selectors, because they allow you to target elements that could never have an ID or class added to them to begin with.
Targeting Specific Elements Without Using IDs or Classes

- Pseudo-classes and pseudo-elements as a whole are not new, or particular, to CSS3.

- However, CSS3 added several individual pseudo-classes that allow us even more precise control over what parts of the document we want to target.

- Many of these new selectors are *structural* pseudo-classes.
What’s the Difference between a Pseudo-Class and a Pseudo-Element?

- Pseudo-classes select HTML elements that could have classes added to them, while pseudo-elements select things that aren’t HTML elements at all.

- The four pseudo-elements in CSS are:
  - ::first-line
  - ::first-letter
  - ::before
  - ::after

- All of these are fragments of other HTML elements, not individual elements themselves. They’re not part of the document tree, so the only way to target them is with pseudo-element selectors.
What’s the Difference between a Pseudo-Class and a Pseudo-Element?

- In terms of syntax, in CSS3, pseudo-classes start with one colon and pseudo-elements start with two. (They used to both have one, and this syntax still works.)

- You can have only **one pseudo-element per selector**, and it has to come at the end (E.g., `#article p::first-line`);

- Pseudo-classes don’t have these restrictions.
CSS3 introduces the concept of "structural pseudo-classes" to target elements in the document tree based on unique characteristics of the elements, such as relative placement.

E.g., the :first-child pseudo-class targets an element that is the first child element of its parent element.

This child element is a standalone HTML element in the document tree, but what makes it unique is that it’s first, and it’s this unique characteristic that we want to be able to select by, without having to add a class or ID.
New Structural Pseudo-classes

- All of the structural pseudo-classes are based on the document object model (DOM).

- To recap, the DOM is the hierarchical structure of the HTML page, made up of elements, attributes, and text, each called a node.
New Structural Pseudo-classes

- It contains multiple levels because elements are nested inside each other.

- Elements nested directly inside other elements are called children of those outer elements; they’re also descendants, along with elements that are nested further down.

- The outer elements are called parents (if one level up) or ancestors (if two or more levels up).

- Elements that are nested at the same level with each other - in other words, they have the same parent - are called siblings.
New Structural Pseudo-classes

- An element can be many or all of these things at once, just like you can be someone’s child and someone else’s parent at the same time.

- The terms are all relative to where a certain element is in relation to a certain other element.

- Other than the :first-child pseudo-class, which is part of CSS 2.1, all of the following structural pseudo-classes are new to CSS3. They offer us a whole host of new ways to target elements very precisely.
New Structural Pseudo-classes

<table>
<thead>
<tr>
<th>PSEUDO-CLASS</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>:root</td>
<td>Selects the element that is the root of the document. In HTML, this is always the html element.</td>
</tr>
<tr>
<td>:nth-child()</td>
<td>Selects based on position within the list of its parent's children.</td>
</tr>
<tr>
<td>:nth-last-child()</td>
<td>Same as :nth-child(), but the counting for the position number is done from the last child upward instead of the first child downward.</td>
</tr>
<tr>
<td>:nth-of-type()</td>
<td>Selects based on position within the list of its parent's children, but only counting children of a certain type (such as p, img, etc.).</td>
</tr>
<tr>
<td>:nth-last-of-type()</td>
<td>Same as :nth-of-type(), but counting from the last child of the specified type instead of the first.</td>
</tr>
<tr>
<td>:first-child</td>
<td>Selects the first child of a parent element.</td>
</tr>
<tr>
<td>:last-child</td>
<td>Selects the last child of a parent element.</td>
</tr>
<tr>
<td>:first-of-type</td>
<td>Selects the first sibling of its own type in a parent element.</td>
</tr>
<tr>
<td>:last-of-type</td>
<td>Selects the last sibling of its own type in a parent element.</td>
</tr>
<tr>
<td>:only-child</td>
<td>Selects an element that is the only child of its parent.</td>
</tr>
<tr>
<td>:only-of-type</td>
<td>Selects the only element of its own type in the parent element.</td>
</tr>
<tr>
<td>:empty</td>
<td>Selects elements that have no children elements or text inside them.</td>
</tr>
</tbody>
</table>
How :nth-child() Works

- :nth-child() selects an element based on its position within the list of its parent’s children; in other words, it selects an element based on how many siblings it has before it.

- You write the position number of the element you want to select inside the parentheses of the selector.

- E.g. li:nth-child(5) would match the fifth li element in a list.

- In addition to numbers inside the parentheses (the selector’s argument), you can also use the keyword odd or even to select every other element in a row, such as the second, fourth, sixth, and so forth.
How :nth-child() Works

- But where :nth-child() gets really powerful is when you use a formula as its argument.

- This allows you to create more complex alternating patterns or even select specific blocks of sequential children at a time.

- The formula has the syntax $an+b$, where $a$ is a cycle size that you pick, $n$ is a counter starting at zero, and $b$ is an offset value that you pick.
How \( \text{nth-child()} \) Works

- Here’s an example: \( \text{li:nth-child}(3n+1) \)

- Since \( n \) starts at zero and then increases by one each cycle, this selector would match:
  - \( (3 \times 0) + 1 = 1 = 1 \text{st list item} \)
  - \( (3 \times 1) + 1 = 4 = 4\text{th list item} \)
  - \( (3 \times 2) + 1 = 7 = 7\text{th list item} \)
  - \( (3 \times 3) + 1 = 10 = 10\text{th list item} \)

- And so on!
How :nth-child() Works

- While you could certainly add classes to the first, fourth, seventh, and tenth list items, it’s time-consuming to do so, easy to forget to do, adds to the weight of your pages, and - probably most importantly - is a pain to maintain.

- If you ever want to add another list item in between existing ones, you have to re-class all the list items from that point forward, as their position numbers will have all changed.

- Using the :nth-child() pseudo-class that keeps track of the position numbers for you and matches accordingly is far more efficient.
How :nth-child() Works

- Don’t let the maths scare you off from using :nth-child(). There are some great online tools that can help you get a better sense for how :nth-child() works by letting you play around with values to see how they affect the styles of the page immediately.

Zebra Striping

- One of the most ubiquitous uses of :nth-child() is to make every other row of a table a different colour; commonly called "zebra striping."

- It can often be more than just an aesthetic enhancement; it can increase usability by making it easier to scan across a long table without losing your place.

- Without :nth-child(), you zebra stripe a table by applying a class to every other row, called something like "even" or "alt", and then give this class a different background colour.

- You have to either apply these classes manually or have a piece of JavaScript do it for you. Neither solution is as efficient as :nth-child().
Zebra Striping

- You can use `:nth-child()` formulas to zebra stripe; the formula `2n` would match all the even rows.

- But the keywords `even` and `odd` are shortcuts that are easier to use.

- We’ll use the `even` keyword in our blog comments page to make every other speech bubble a different colour.
Zebra Striping

- Right now, all the speech bubbles in this page are the same shade of greenish-blue. This colour has the value \texttt{hsla(182,44\%,76\%,.5)}.

- Let's use a bluer shade for our alternating colour.

- Add this new rule to the CSS:
  \texttt{li:nth-child(even) blockquote { background-color: hsla(250,70\%,82\%,.5); }}

- You’ll see that the second and fourth comments are purpleish-blue, while the first and third are still greenish-blue.
Let’s return to the homework page we worked on previously to see how we can achieve alternating styles on the images within the page.

Right now, all the images are rotated in order to make them appear more realistic. But since they’re all rotated the same amount, they look very uniform.

It would be nice to be able to use :nth-child() to rotate different photos different amounts to enhance the appearance of randomness and realism.
Random Rotation

- However, if you used the selector `img[src*='photos'] :nth-child(even)` to rotate all the even-numbered images to the left instead of the right, you might be surprised to find that the last two images both rotate right, instead of alternating.

- This is because the `:nth-child()` pseudo-class selects all children of the same parent.

- The `img` elements are siblings with all the `p` and `h2` elements, so all of these elements are counted for `:nth-child()`.
Random Rotation

- Even though img is included in the selector, all that the selector is saying is "Find all the images that have ‘photos’ in their src attribute. Then apply these styles to the ones that are even-numbered children."

- If you count all the img, p, and h2 elements in the parent div, you’ll find that the second-to-last photo is child number 29 of the div, and the last photo is child number 37.

- Thus, the :nth-child() rule selecting even-numbered children doesn’t apply to either of them, and they stay rotated to the right.
What we really need is a selector like :nth-child() but that counts only elements of a particular type.

Lucky for us, CSS3 provides such a selector: the :nth-of-type() pseudo-class.

It works exactly the same as :nth-child(), but it counts only whatever element you specify in front of it.
Random Rotation

- Add the following new rule to the styles:

```css
img[src*=photos]:nth-of-type(even) {
  -moz-transform: rotate(-2deg);
  -o-transform: rotate(-2deg);
  -webkit-transform: rotate(-2deg);
  transform: rotate(-2deg);
}
```
Random Rotation

- Notice that the first, third, and fifth photos are rotated to the left, even though the selector says to rotate even numbered ones to the left.

- That's because there's another `img` element before all the photos on the page: the calendar thumbnail. This `img` element makes the first, third, and fifth photos the second, fourth, and sixth images overall.

- The `:nth-of-type()` pseudo-class only cares about the element type when doing its counting - in this case, that element type is `img`.

- What the full selector is saying is "Find all the images that have 'photos' in their `src` attribute. Then apply these styles to those that are `even-numbered img` element children."
There’s no way in CSS3 to make the browser count only `img` elements that have particular attributes.

Any other `img` elements mixed in with the photos are going to be used for counting and calculating the child number.

In the case of our page, we’re just trying to make the photos look random, so having other images interrupt our pattern isn’t a bad thing.

The `:nth-of-type()` pseudo-class works for our purposes, even if it can’t select exactly what we might like.
Dynamically Highlighting Page Sections

- You’ve now seen two examples of how CSS3’s structural pseudo classes can add visual enhancements to your pages while keeping your code free of classes and IDs, and without using JavaScript.

- Other CSS3 pseudo-classes can also add much more dynamic-looking effects to your pages, such as highlighting the current section when you use a within-page link to jump down the page.

- This is not only a visual enhancement, but a usability one, as it helps orient the viewer to where they are in the page.
The :target Pseudo-class

- Some URLs have fragment identifiers, represented by the character # followed by an anchor name or element ID, to link to a certain element in the page. E.g. http://en.wikipedia.org/wiki/Jane_austen#cite_note-21.

- The :target pseudo-class selects the element being linked to, and lets you style it.
Adding the Table of Contents

- Right now, the homework page doesn't have any fragment identifiers we can link to. Let's add IDs to all of the subheads in the page, since they naturally divide it up into sections.

- In your page, add `id` attributes to each `h2` element, starting with the "January" one, with the values shown:
  
  ```html
  <h2 id="january">January</h2>
  <h2 id="february">February</h2>
  <h2 id="march">March</h2>
  <h2 id="april">April</h2>
  <h2 id="may">May</h2>
  <h2 id="june">June</h2>
  ```
Adding the Table of Contents

- Now add a table of contents to the top of the page that will link to each of these h2 elements.

```html
<ul id="toc">
  <li><a href="#january">January</a></li>
  <li><a href="#february">February</a></li>
  <li><a href="#march">March</a></li>
  <li><a href="#april">April</a></li>
  <li><a href="#may">May</a></li>
  <li><a href="#june">June</a></li>
</ul>
```
Changing Background Colour on the Jumped-to Section

- We want to highlight the section of the page that you jump to when you click one of the links in the table of contents.

- The element that is targeted when you click a link is an h2 element, so the selector we need is \texttt{h2:target}.

- Create a new rule with this selector, and assign it a background colour of the same shade of blue used for the number icons, but at a more semitransparent level:

\begin{verbatim}
\texttt{h2:target} { 
\texttt{background-color: hsla(203,78\%,36\%,.2);} 
}
\end{verbatim}
Changing Background Colour on the Jumped-to Section

- To spruce up the appearance a bit, you can add some left padding and a shadow to the text:

```html
h2:target {
  padding-left: 10px;
  background-color: hsla(203, 78%, 36%, .2);
  text-shadow: 1px 1px 2px #fff;
}
```
More on the :nth-child() Pseudo-Class

- The :nth-child() pseudo-class is part of the Selectors module found at www.w3.org/TR/css3-selectors.

- It’s a structural pseudo-class that selects an element based on how many siblings precede it within the same parent element.

- Inside the parentheses of :nth-child(), you write either
  - a number (to select one particular child),
  - the keyword **odd** or **even** (to select every other child, either odd-numbered or even-numbered), or
  - a formula in the syntax an+b (to select a particular combination of children you want). In this formula, a is a cycle size, n is a counter that starts at zero, and b is an offset value.
More on the :\texttt{nth-child()} Pseudo-Class

- Negative values are allowed for \(a\) and \(b\).

- If \(a\) is 1, you can omit it (so \(1n+3\) is the same as \(n+3\)).

- If \(b\) is 0, or if \(a\) and \(b\) are equal, you can omit the \(b\) value (so \(2n+0\) and \(2n+2\) are the same as \(2n\)).

More on the :nth-child() Pseudo-Class

- Other than zebra striping, you might want to use :nth-child() for:
  - Styling the first two or more paragraphs of an article differently (using -n+2, if styling just the first two).
  - Giving the first ten items in a top-100 list a larger font size (using -n+10).
  - Making older blog posts or Tweets in a list have a smaller font size or fainter colour as you move down the list.
  - Styling specific table columns differently (E.g., making the third column, which contains numbers, have right-aligned text).
More on the :nth-child() Pseudo-Class

- Creating the appearance of randomness (E.g., making every third feature box have one background colour, every fourth have another, and so on).

- Forcing a line break or margin change at every fourth image thumbnail, E.g., to create an image gallery with multiple rows of thumbnails all in the same HTML list; see http://mondaybynoon.com/2010/03/18/css3-center-thumbnail-galleries.

- Changing the width of side-by-side items based on how many are there, to always fill the available space; see http://andr3.net/blog/post/142.
### :nth-child() Browser Support

<table>
<thead>
<tr>
<th>Selector</th>
<th>Chrome</th>
<th>Edge</th>
<th>Firefox</th>
<th>Opera</th>
<th>Safari</th>
</tr>
</thead>
<tbody>
<tr>
<td>:nth-child()</td>
<td>4.0</td>
<td>9.0</td>
<td>3.5</td>
<td>3.2</td>
<td>9.6</td>
</tr>
</tbody>
</table>
More on the :nth-of-type() Pseudo-class

- The :nth-of-type() pseudo-class is part of the Selectors module found at [www.w3.org/TR/css3-selectors](http://www.w3.org/TR/css3-selectors).

- It’s a structural pseudo-class that selects an element based on how many siblings of the same type come before it within the same parent element.

- It takes the same sorts of values for its argument (inside the parentheses) as :nth-child().
More on the :nth-of-type() Pseudo-class

- Other than rotating photos, you might want to use :nth-of-type() for:
  - Creating the appearance of randomness in some way other than varying the rotation.

- Alternating images within an article floating left and right.

- Styling the first one or more paragraphs of an article differently; (if other elements might prevent those paragraphs from reliably being the first children, such as an h2 or img that sometimes comes first, :nth-child() won’t work)
More on the \texttt{:nth-of-type()} Pseudo-class

- Alternating styles on terms within a definition list; since each \texttt{dt} element may have only one or multiple \texttt{dd} elements following it, you can’t use \texttt{:nth-child()}.  

- Alternating styles on \texttt{blockquote} elements within an article.
### nth-of-type() browser support

<table>
<thead>
<tr>
<th>Selector</th>
<th>Chrome</th>
<th>Edge</th>
<th>Firefox</th>
<th>Safari</th>
<th>Opera</th>
</tr>
</thead>
<tbody>
<tr>
<td>nth-of-type()</td>
<td>4.0</td>
<td>9.0</td>
<td>3.5</td>
<td>3.2</td>
<td>9.6</td>
</tr>
</tbody>
</table>
More on the :target Pseudo-Class

- The :target pseudo-class is part of the Selectors module found at www.w3.org/TR/css3-selectors.

- It allows you to select an element that is the target of a referring URL with a fragment identifier in it.
More on the :target Pseudo-Class

- Other than highlighting the heading of the current page section, you might want to use it for:
  - Highlighting footnotes

- Revealing explanatory text next to a targeted heading, so the user gets more context for where she is in the page; see http://web-graphics.com/mtarchive/001454.php

- Bringing an item to the front of a stack of overlapping boxes or images; see http://virtuelvis.com/archives/2003/07/target-fun

- Tabbed content boxes; see http://css-tricks.com/css3-tabs
More on the :target Pseudo-Class

- Accordion menus or expanding and collapsing content boxes; see www.paulrhayes.com/2009-06/accordion-using-only-css and www.thecssninja.com/css/accordion-effect-using-css


- Modal windows or lightboxes; see http://sixrevisions.com/css/semantic-css3-lightboxes and www.thecssninja.com/css/futurebox2
More on the :target Pseudo-Class

- Please note that some of these techniques are probably better controlled with JavaScript than CSS, due to potential accessibility and usability problems with pure CSS versions.

- That said, they might be useful in certain limited circumstances or provide you with ideas for other ways to use :target effectively.
### :target Browser Support

<table>
<thead>
<tr>
<th>Selector</th>
<th>Chrome</th>
<th>Edge</th>
<th>Firefox</th>
<th>Safari</th>
<th>Opera</th>
</tr>
</thead>
<tbody>
<tr>
<td>:target</td>
<td>4.0</td>
<td>9.0</td>
<td>3.5</td>
<td>3.2</td>
<td>9.6</td>
</tr>
</tbody>
</table>