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
- Transitions to change the value of a property gradually instead of abruptly
- Animations to control more complex visual changes
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.
New Structural Pseudo-classes

- 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>
We can use the :nth-child() pseudo-class to make every other speech bubble in our comments page have a different background colour.

And we’ll do this without using classes or JavaScript!
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 \texttt{\textasciicircum{nth-child}()} Works

- But where \texttt{\textasciicircum{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 \texttt{nth-child()} Works

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

- Since \textit{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 \texttt{\textbf{\text:{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 \texttt{\textbf{\text:{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.
Zebrato Striping

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

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

- Add this new rule to the CSS:
  ```css
  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.
Random Rotation

- 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.
Random Rotation

- 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.”
Random Rotation

- 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.
In fact, let’s make the photos look even more random by adding another `:nth-of-type()` rule:

```css
img[src*='photos'] :nth-of-type(3n) {
  -moz-transform: rotate(1deg);
  -o-transform: rotate(1deg);
  -webkit-transform: rotate(1deg);
  transform: rotate(1deg);
}
```
Random Rotation

- This makes every third image angled to the right by one degree.

- The photos have a fairly random-looking pattern of rotation now: the first is rotated negative -2deg, the second 1deg, the third negative -2deg, the fourth 2deg, and the fifth 1deg.

- Even though the `:nth-of-type()` selector may not do exactly what you expect and want, it still provides a heap of control over what elements you want to target without having to resort to classes or IDs.
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>
```
Styling the Table of Contents

- The table of contents list already has some non-standard styling because of the existing rules for `ul` and `a` elements; the links are all on one line and spaced out from each other. Let’s enhance the styles further.

- Add a background image of a hand-drawn arrow to the list:

```css
#toc {
  background: url(images/arrow.gif) no-repeat top right;
  padding-top: 1.6em;
}
```
Styling the Table of Contents

- Next, get rid of the left padding on the li and a elements and use right padding instead, so the list as a whole is aligned on the left side:

```css
#toc li {
  padding: 0 1.2em 0 0;
}
#toc a {
  padding-left: 0;
}
```
Creating Number “Icons” with Pure CSS

- To create numbers in front of the list items, we could use an ordered list (\texttt{ol} element) instead of an unordered list (\texttt{ul} element).

- However, there’s no way to directly style the list marker numbers that the browser adds.

- There are ways to hack around this, but they limit the looks we can achieve and add junk to the markup.
Creating Number “Icons” with Pure CSS

- Another option is to use background images of numbers. This has the disadvantage, though, of adding five more HTTP requests to the page.

- Instead of using images, let’s use generated content like we did before to insert the numbers for us.

- But we’ll take it a step further. Instead of hard-coding the actual numbers in the content property - which would require five different rules for the five different list items - we’ll use CSS counters, a CSS 2.1 feature, to dynamically generate and increment the numbers.
To use counters, you must first create a set of counters with a name of your choosing, using the `counter-reset` property:

```css
#toc {
  background: url(images/line.png) no-repeat top right;
  padding-top: 1.6em;
  counter-reset: list;
}
```
This establishes a set of counters, arbitrarily named “list”, that you can now apply to a sequence of elements. (You can also set a base value to start counting from in the counter-reset property, but it’s zero by default, which is what I want, so I haven’t included a number here.)

The elements we want to apply the “list” set of counters to are the sequence of li elements inside the table of contents list.
Creating Number “Icons” with Pure CSS

- To apply the counters, use the `counter-increment` property in the `#toc li` rule:

```css
#toc li {
  padding: 0 1.2em 0 0;
  counter-increment: list;
}
```
Creating Number “Icons” with Pure CSS

- This tells the browser that you want to increment the counter on each li element, but it doesn’t actually display the counter. You need to use the `content` property to do that.

- Create a new rule using the `:before` pseudo-class on the li elements to make the counters display before each list item’s content:

```css
#toc li:before {
  content: counter(list);
}
```
Creating Number “Icons” with Pure CSS

- This tells the browser that the content you want to display is a counter, and the name of that counter is “list.”

- And with that, the numbers magically appear before each list item, starting at one and incrementing by one on each new list item.
Creating Number “Icons” with Pure CSS

- We can style these numbers just like any other pieces of content in our pages.

- First, let’s get them on the same line as the text, by floating both the numbers and text and adding a little left padding to the list items:

  ```css
  #toc li:before {
  content: counter(list);
  float: left;
}
#toc a {
  float: left;
padding-left: 5px;
}  ```
Creating Number “Icons” with Pure CSS

- Now let’s give each number a circular background using `border-radius`, in the same shade of blue as the links, but semitransparent:

```css
#toc li:before {
  content: counter(list);
  float: left;
  width: 1.6em;
  height: 1.6em;
  -moz-border-radius: .8em;
  -webkit-border-radius: .8em;
  border-radius: .8em;
  background: #87B3CE;
  background: hsla(203,78%,36%,.5);
}
```
Creating Number “Icons” with Pure CSS

- Now the text needs some further alignment within those circles.

- Add these new declarations to the `#toc li:before` rule:
  ```css
color: #fff;
font-family: Arial, Helvetica, “Helvetica Neue”, sans-serif;
font-weight: bold;
text-decoration: none;
text-shadow: 0 1px 0 hsla(0,0%,0%,.6);
text-align: center;
```
Creating Number “Icons” with Pure CSS

- Browsers that don’t understand generated content will not see the numbers, let alone their styles.

- In this case, the numbers are decorative, not essential content, so this is an acceptable instance of progressive enhancement.
Changing Background Colour on the Jumped-to Section

- All of this work on the table of contents was just a prelude to what we really came here to do: 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 `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:

```css
h2:target {
  background-color: hsla(203, 78%, 36%, .2);
}
```
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:

```css
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](http://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 :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).

- For more details on this, see http://reference.sitepoint.com/css/understandingnthchildefxpressions.
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>Safari</th>
<th>Opera</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.

- 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 :nth-of-type() Pseudo-class

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

- Alternating styles on `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/accordian-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>