adtrak

CSS Selectors

Stephen Greig

‹a›

CSS Selectors

Introduction

This will be a comprehensive run through of CSS selectors, showing how we can use them creatively and combine them to target elements (or particular states of elements) when targeting via a class isn't appropriate or necessary.

- Basic Selectors & "Combinators"
- Attribute Selectors
- Pseudo-elements
- Pseudo-classes
  - Form & validation
  - Child pseudo-classes
  - nth-child expressions

Basic Selectors & Combinators

Your Basic Toolkit

div .class #id

* > + ~

Basic Selectors & Combinators

The Universal Selector (*)

- Selects ALL elements
- Often found at the top of stylesheets to set universal styles (such as `box-sizing`)
- It's sometimes forgotten that it can be used to target all elements within a containing element as well

```
* {
  box-sizing: border-box;
}
```

```
.sidebar * {
  margin-top: 20px;
}
```

Basic Selectors & Combinators

The Child Combinator (>)

- Selects only direct children of a parent container
- For example, in a multi-level nav menu, you can target only the top level `<li>`s without the styles cascading to sub menu `<li>`s
- Supported in IE7+

```
.main-nav > li { margin-top: 20px; }
```

```
<ul class="main-nav">
  <li>One</li>
  <li>Two
    <ul>
      <li>Sub Item</li>
    </ul>
  </li>
  <li>Three</li>
</ul>
```

The Adjacent Sibling Combinator (+)

- Selects an element that **immediately** follows another element
- Supported in IE7+
- Useful for things like nav item borders

```
h1 + p { font-weight: bold; }
```

```
<h1>Headline</h1>
<p>Lorem ipsum</p>
<p>Lorem ipsum</p>
<p>Lorem ipsum</p>
```


```
li + li { border-left: 1px solid #000; }
```

Basic Selectors & Combinators

The General Sibling Combinator (~)

- Selects elements that follow another element but not necessarily immediately
- Supported in IE7+

```
h1 ~ p { font-weight: bold; }
```

```
<h1>Headline</h1>
<p>Lorem ipsum</p>
<div>Dolor sit</div>
<p>Lorem ipsum</p>
<p>Lorem ipsum</p>
```

Combining Basic Selectors & Combinators

- These combinators can be used creatively to solve common issues easily

```
.sidebar > * + * { margin-top: 20px; }
```

- This would target any direct child that follows another direct child of the sidebar.
- So it's an easy way of implementing consistent margins

Attribute Selectors

CSS Selectors

Attribute Selectors

- Target through HTML attributes
- Supported in IE7+

Targets elements with an **exact attribute value**

```
input[type="submit"]
```

Targets elements if the attribute value **contains** a certain string

```
a[href*="twitter"]
```

Targets elements if the **attribute has been applied**

```
button[disabled]
```

Targets elements if the attribute value **ends** with a certain string

```
a[href$=".de"]
```

Targets elements if the attribute value **starts** with a certain string

```
a[href^="https://"]
```

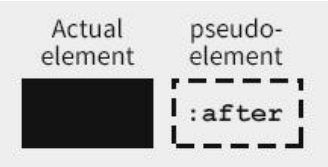
Pseudo-elements & Pseudo-classes

Pseudo-elements and Pseudo-classes

What's the difference?

- **Pseudo-classes** provide ways of targeting actual, existing elements

- **Pseudo-elements** don't target actual elements; they do not exist in the HTML



- Officially differentiated by using a double colon for pseudo-elements (**::after**)
- And a single colon for pseudo-classes (**:first-child**)
- But some older browsers (cough, IE) decided to implement support for pseudo-elements but only using a single colon. So single colons have generally been used ever since for that added backward compatibility.

Pseudo-elements

Pseudo-elements

- **:before** (IE8+)
- **:after** (IE8+)
- **:first-line** (IE5.5+)
- **:first-letter** (IE5.5+)

Newer/unofficial pseudo-elements (these require the double colon)

- **::selection** (IE9+)
- **::placeholder** (IE10+)

Pseudo-elements

:before and :after

- Most of you have used these before
- They can be styled just like a real element, but they do require the content property (although this can be left blank using empty quotation marks: `content: "";`)

- A little known trick is that you can use them to output an element's attribute value
- This example would show an <a> element's href value after it, which could be useful for print stylesheets

```
a:after {
  content: " ("attr(href)")";
}
```

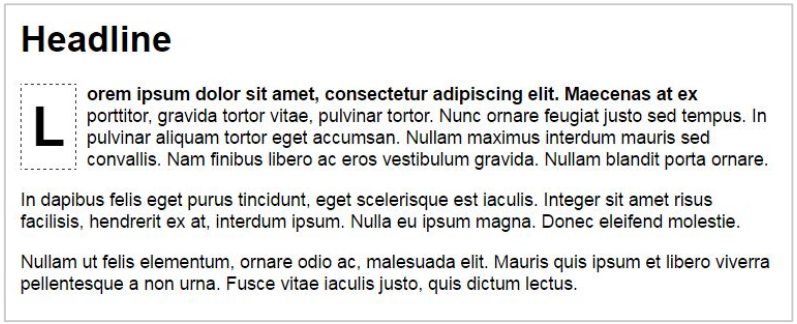
```
<a href="http://adtrak.co.uk">Adtrak</a>
```

Adtrak (http://adtrak.co.uk)

Pseudo-elements

:first-line and :first-letter

- Predictably, these pseudo-elements provide a way of styling an element's first line of text and/or its first letter
- Appending these pseudo-elements onto `:first-of-type` ensures that they only affect the first paragraph, as show below




```
p:first-of-type:first-line {
 font-weight: bold;
}
```

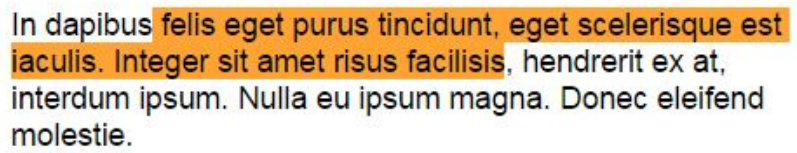
```
p:first-of-type:first-letter {
  font-size: 3em;
  padding: 10px;
  border: 1px dashed #333;
  float: left;
  margin: 0 10px 10px 0;
}
```

Pseudo-elements

::selection

- Not in the official spec but does have very good browser support
- Requires the double colon
- Allows you to control the styling of your selected text

```
p::-moz-selection,
p::selection {
  background: orange;
}
```



Pseudo-elements

::placeholder

- Not in the official spec but does have very good browser support (IE10+), although the implementation in the various browsers differs slightly
- Requires the double colon
- Allows you to style input placeholder text

```
input::-webkit-input-placeholder,
input::-ms-input-placeholder,
input::-moz-placeholder {
  color: orange;
}
```

Pseudo-classes

Pseudo-classes

State Based Pseudo-classes

- :link
- :hover
- :visited
- :active
- :focus

Form & Validation

- :checked
- :default
- :disabled
- :enabled
- :in-range
- :out-of-range
- :indeterminate
- :valid
- :invalid
- :optional
- :read-only
- :read-write
- :required

Child Pseudo-classes

- :first-child
- :first-of-type
- :last-child
- :last-of-type
- :only-child
- :only-of-type
- :nth-child
- :nth-of-type
- :nth-last-child
- :nth-last-of-type

Other

- :empty
- :root
- :lang
- :not
- :target

Pseudo-classes

:not - The Negation Pseudo-class

- `:not` allows you to select an element, except for a specified variation of that element

```
li:not(:first-child) {
  background: orange;
}
```

This would select all `li`s except the first


```
body:not(.home) {
  background: orange;
}
```

This would select the body element, unless it has a class of “home”

Pseudo-classes

:empty

```
<ul>
  <li>ABC</li>
  <li>DEF</li>
  <li></li>
  <li>JKL<li>
  <li> <li>
</ul>
```

```
li:empty {
  display: none;
}
```

- This pseudo-class allows you to target completely empty elements
- Note that the 3rd `li` in this example would be targeted, but not the 5th
- The element must be completely empty, including spaces

Pseudo-classes

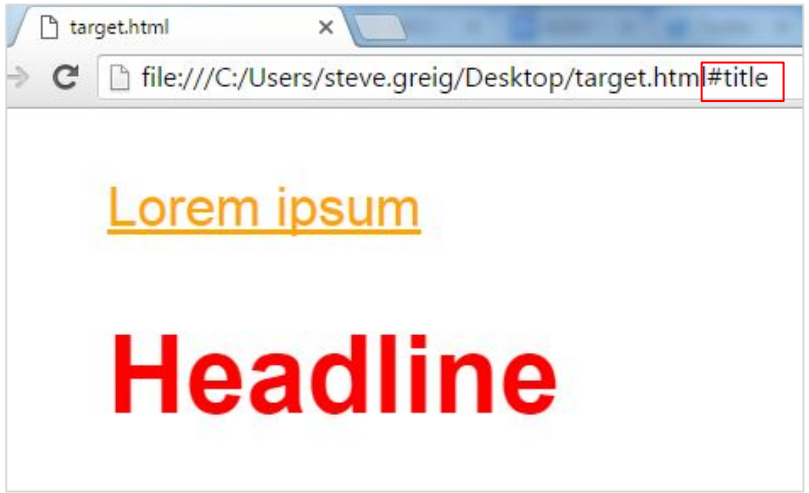
:target

- The `:target` pseudo-class provides a way of triggering style changes on click, without any JS
- It does this using the “url fragment identifier”

```
<a href=”#title”>Lorem ipsum</a>

<h1 id=”title”>Headline</h1>
```

```
h1:target {
  color: red;
}
```



Pseudo-classes

Getting Creative with :target

- This functionality can be quite powerful…

Demo: http://codepen.io/stephengreig/pen/JXzMPP?editors=1100

Pseudo-classes

:checked

- Applies to radio inputs, checkbox inputs and select menu options
- Allows you to add styles to the currently checked/selected item

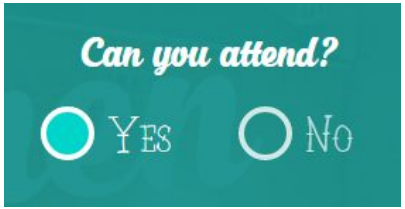
```
<input type="radio" id="abc">
<label for="abc">ABC</label>
```

ABC
XYZ

```
input:checked + label {
  color: red;
}
```

ABC
XYZ

- Opens the door for custom form controls...



Pseudo-classes

The “Checkbox Hack”

- You might have heard of this term or even used this technique before
- It basically uses the `:checked` pseudo-class to create some kind of click-based functionality, essentially expanding on the idea in the previous example

```
<input type=”radio” id=”abc”>
<label for=”abc”>Lorem</label>

<div>Just some text</div>
```

```
div { display: none; }

input:checked ~ div {
  display: block;
}
```

Demo: http://codepen.io/stephengreig/pen/grJbeW?editors=1100

Child Pseudo-classes

Child Pseudo-classes

Basic Child Pseudo-classes

- **:first-child**
- **:first-of-type**
- **:last-child**
- **:last-of-type**
- **:only-child**
- **:only-of-type**

```
<div>
  <p>One</p>
  <p>Two</p>
  <p>Three</p>
</div>
```

```
p:first-child {
  color: red;
}
```

```
p:last-child {
  color: green;
}
```

Something that often trips people up...

```
<div>
  <h1>Title</h1>
  <p>One</p>
  <p>Two</p>
  <p>Three</p>
  <ul>...</ul>
</div>
```

The `<h1>` is now the first child and the `<ul>` is the last child

```
p:first-of-type
```

```
p:last-of-type
```

Child Pseudo-classes

Another Thing to Note

Similarly, you need to bear in mind that classes have no bearing with child pseudo-classes, which is a bit counter-intuitive so can often trip people up

```
<div>
  <p class=”one”></p>
  <p class=”two”></p>
  <p class=”two”></p>
</div>
```

`.two:first-of-type` will not work, simply because it is not the first `<p>` element (the class is irrelevant)

nth-child Pseudo-classes

nth-child Pseudo-classes

`p:nth-child(2)`	`p:nth-of-type(2)`	`p:nth-last-child(2)`	`p:nth-last-of-type(2)`
Selects the 2nd child	Selects the 2nd `<p>` element	Selects the 2nd child, counting from the end	Selects the 2nd `<p>` element, counting from the end

```
<div>
  <h1>Title</h1>
  <p>1</p>
  <p>2</p>
  <p>3</p>
  <ul>...</ul>
</div>
```

The p can be omitted here as the 2nd child is selected regardless of type

```
<div>
  <h1>Title</h1>
  <p>1</p>
  <p>2</p>
  <p>3</p>
  <ul>...</ul>
</div>
```

```
<div>
  <h1>Title</h1>
  <p>1</p>
  <p>2</p>
  <p>3</p>
  <ul>...</ul>
</div>
```

The p can be omitted here as the 2nd last child is selected regardless of type

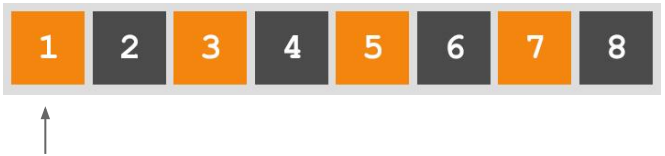
```
<div>
  <h1>Title</h1>
  <p>1</p>
  <p>2</p>
  <p>3</p>
  <ul>...</ul>
</div>
```

nth-child Expressions

nth-child Expressions

`:nth-child(2n)`

`:nth-child(2n+1)`



An easy way to remember how this works is: the first number states the sequence (so every 2nd element)
And the second number states where this sequence should start (the 1st element)

To get a better understanding, we just need to break down the expression to see what's going on:

2n+1 is the same as: (2xn)+1

(2xn)+1
(2x0)+1 = **1** (select the 1st element)
(2x1)+1 = **3** (select the 3rd element)
(2x2)+1 = **5** (select the 5th element)
(2x3)+1 = **7** (select the 7th element)
etc.

nth-child Expressions

Another Example

```
:nth-child(3n+4)
```

3n+4 is the same as: (3xn)+4

(3xn)+4
(3x0)+4 = **4** (select the 4th element)
(3x1)+4 = **7** (select the 7th element)
(3x2)+4 = **10** (select the 10th element)
(3x3)+4 = **13** (select the 13th element)
etc.

nth-child Expressions

Using Minus Numbers With nth-child

- Using minus numbers in an nth-child expression allows your sequence to run backwards from a certain point
- So you can use minus numbers to select the first x amount of items...

`:nth-child(-1n+3)`
Can be written as `:nth-child(-n+3)`

-n+3 is the same as: (-1xn)+3

(-1xn)+3
(-1x0)+3 = **3** (select the 3rd element)
(-1x1)+3 = **2** (select the 2nd element)
(-1x2)+3 = **1** (select the 1st element)
(-1x3)+3 = **0** (nothing to select)
(-1x4)+3 = **-1** (nothing to select)
etc.

nth-child Expressions

Combining Expressions to Target Ranges

You can combine two different nth-child pseudo-classes to select an isolated range of items

`:nth-child(n+3)` would select every item starting from the 3rd

`:nth-child(-n+6)` would select every item (counting backwards) starting from the 6th (so the first 6 items)

So if you combine them together, it will target items 3-6

`:nth-child(n+3):nth-child(-n+6)`

:nth-last-child Expressions

All of the examples from the last few slides work the same with **`:nth-last-child`** and **`:nth-last-of-type`**, except that **they count from the end**, instead of the start

`:nth-last-child(n+3)`:`nth-last-child(-n+6)`

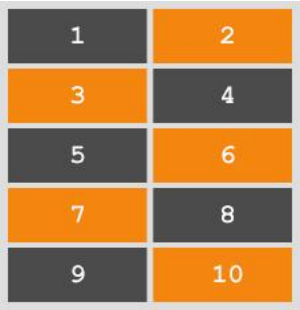
`:nth-last-child(-n+6)`

`:nth-last-child(2n+5)`

nth-child Expressions

Multiple Expressions for more Abstract Patterns

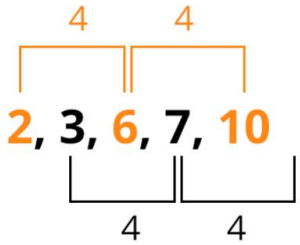
You will often have the need to select elements in a more abstract pattern, rather than a simple sequence of every 2 or every 3 items. For example...



This is a very common design pattern, but there isn't an obvious way to select these elements using a simple nth-child expression as they don't represent a simple even sequence.

Instead, the items we need to target are: **2, 3, 6, 7, 10**

To get around this, we need to analyse the pattern and pull out the even sequences that do exist, and then we can simply target them both…

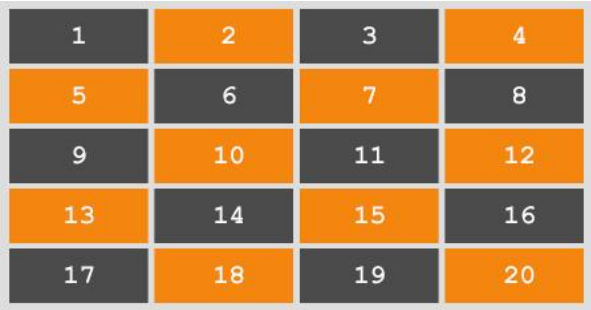



```
nth-child(4n+2),
nth-child(4n+3) {
  background: orange;
}
```

nth-child Expressions

Multiple Expressions for more Abstract Patterns

If we expand the previous example to a 4-column grid, it becomes a bit more complex



This time, the items we need to target are:
2, 4, 5, 7, 10, 12, 13, 15, 18, 20

It's not immediately obvious looking at those numbers, but there are 4 separate even sequences going on, which are more evident if you look at each column in the grid…

2, 4, 5, 7, **10**, 12, 13, 15, **18**, 20 = Every 8th item starting from the 2nd
2, **4**, 5, 7, 10, **12**, 13, 15, 18, **20** = Every 8th item starting from the 4th
2, 4, **5**, 7, 10, 12, **13**, 15, 18, 20 = Every 8th item starting from the 5th
2, 4, 5, **7**, 10, 12, 13, **15**, 18, 20 = Every 8th item starting from the 7th

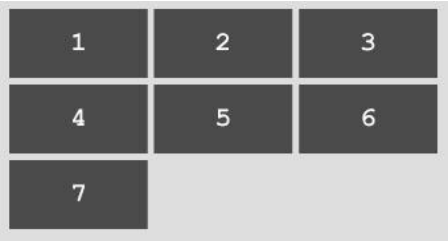
```
:nth-child(8n+2),
:nth-child(8n+4),
:nth-child(8n+5),
:nth-child(8n+7) {
  background: orange;
}
```

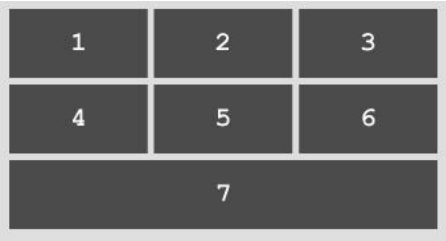
CSS
If Statements?!

CSS If Statements?!

Targeting the Last Item if Total Items = x

- You may be surprised to learn that we can also use CSS selectors to style elements based on certain conditions, almost like an if statement!
- For example, if you have a 3-column grid, you might want to say:
- **If there are exactly 7 elements, give the last one 100% width**






```
:nth-child(7):last-child {
  width: 100%;
}
```

- By combining 2 pseudo-classes, we can target the last child, but only if it's the 7th element!

CSS If Statements?!

Targeting the Last *Few* Items if Total Items = x

- Alternatively, instead of making that last child full width, you might want to fit it onto the previous row, as shown below
- So now we basically need to target those last 4 items to alter their widths, but we only want to do that if there are 7 in total!

- We target the item that is **4th from the start** *and* **4th from the end**; this can only exist if there are 7 in total
- Then we target every item that follows that

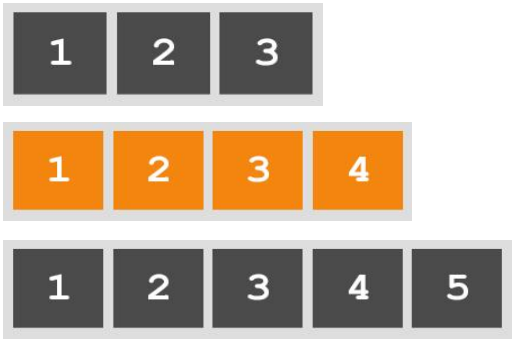
```
li {
  width: 33%;
}

li:nth-child(4):nth-last-child(4),
li:nth-child(4):nth-last-child(4) ~ li {
  width: 25%;
}
```

CSS If Statements?!

Targeting All Items Only If Total = x

- Similarly, you might want to target all of the elements, but only if there are a certain amount in total




```
li {
  background: gray;
}

li:first-child:nth-last-child(4),
li:first-child:nth-last-child(4) ~ li {
  background: orange;
}
```

- We target the item that is both the **first child** *and* the **4th child from the end**; this can only exist if there are 4 items in total
- Then we target every item that follows that

CSS If Statements?!

Targeting All Items Only If Total = at least x

- You're probably more likely to want to apply styles if the total number of items is a certain amount *or greater*

```
li:first-child:nth-last-child(n+6),
li:first-child:nth-last-child(n+6) ~ li {
  background: orange;
}
```

- This one is a bit more complex, so let's break it down...

Target all items if there are 6 or *more* in total

1

`li:nth-last-child(n+6)`
= target every item starting from the 6th (but counting from the end)

2

`li:first-child:nth-last-child(n+6)`
= target every item starting from the 6th (but counting from the end), but only if it's also the first child.

3

`li:first-child:nth-last-child(n+6),`
`li:first-child:nth-last-child(n+6) ~ li`
= target every item starting from the 6th (but counting from the end), but only if it's also the first child.
= then target every item that follows this first-child

CSS If Statements?!

Targeting All Items Only If Total = x or fewer

- Similarly, you might want to apply styles if the total number of items is a certain amount or *less*

```
li:first-child:nth-last-child(-n+6),
li:first-child:nth-last-child(-n+6) ~ li {
  background: orange;
}
```

Target all items if there are 6 or *less* in total

1

`li:nth-last-child(-n+6)`

= target every item starting from the 6th (but counting from the end *and counting backward*)

2

`li:first-child:nth-last-child(-n+6)`

= target every item starting from the 6th (but counting from the end *and counting backward*)

= AND only if it's the first child as well

3

`li:first-child:nth-last-child(-n+6),`
`li:first-child:nth-last-child(-n+6) ~ li`

= target every item starting from the 6th (but counting from the end *and counting backward*)

= AND only if it's the first child as well

= Then target every item that follows this first child

Sass Nesting & Selector Efficiency

Sass Nesting & Selector Efficiency

Avoid Over Nesting

- Sass is essentially a shortcut for actual CSS, not a replacement
- It still compiles to a normal CSS file that should, in theory, be the same as if you wrote that vanilla CSS yourself
- There's a tendency to forget to think about how your Sass code will compile, particularly when it comes to nesting

```
header {
  .main-nav {
    ul {
      li {
        a {
        }
      }
    }
  }
}
```

This will compile as ->

```
header .main-nav ul li a { }
```

Whereas if you were writing vanilla CSS you would write a much more succinct / efficient selector like...

```
.main-nav a { }
```

Sass Nesting & Selector Efficiency

Avoid Over Nesting

```
header {
}

.main-nav {
  ul {
  }

  li {
  }

  a {
  }
}
```

This will compile as...

```
header {}

.main-nav ul {}

.main-nav li {}

.main-nav a {}
```

- Instead, you should only nest where it is sensible to do so
- I generally try to limit nesting to a max of 2 or 3 levels deep
- This keeps your Sass code simple and uncomplicated
- And keeps your CSS selectors short and efficient
- Which is good for performance
- And good for avoiding specificity issues

Sass Nesting & Selector Efficiency

Don’t Over-qualify Your Selectors

- In a similar vein, it’s important not to over-qualify your selectors unnecessarily, i.e...

```
ul.subnav

nav.main-nav

.sidebar .sidebar-links

.my-table tbody tr td
```

Eradicate the unnecessary guff from your selectors

Questions?

Game Time

http://flukeout.github.io/