The Ultimate

CSS SELECTORS

Cheatsheet

BASIC SELECTORS & COMBINATORS

`*`	**The Universal Selector** - select all elements	h1 p p ul p
`h1 + p`	**The Adjacent Sibling Combinator** - select all <p> elements that **immediately** follow a <h1>	h1 p p ul p
`h1 ~ p`	**The General Sibling Combinator** - select all <p> elements that follow (and are siblings of) a <h1>	h1 p p ul p
`.list > li`	**The Child Combinator** - select all elements that are direct children of .list	

```
<ul class="list">
  <li>One ✔
  <li>Two ✔
    <ul>
      <li>Sub Item ✘
```

ATTRIBUTE SELECTORS - Target through HTML Attributes

`button[disabled]`	Target button elements if they have the disabled **attribute applied**
`input[type="submit"]`	Target input elements if they have a type attribute with an **exact** value of submit
`a[href^="http://"]`	Target <a> elements that have a href value that **starts** with http://
`a[href$=".de"]`	Target <a> elements that have a href value that **ends** with .de
`a[href*="twitter"]`	Target <a> elements that have a href value that **contains** the word "twitter"

PSEUDO-ELEMENTS - These target elements that do not exist in the HTML (as opposed to pseudo-classes)

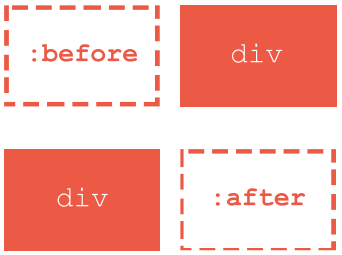
```
div:before {
  content: "";
}

div:after {
  content: "";
}
```

The **:before** and **:after** pseudo-elements allow you to insert content before or after any HTML element that isn't self closing (like <img> and <input>).

The content property is required but can be left blank.

These pseudo-elements can be treated and styled like any other element.



Pseudo-elements continued...

PSEUDO-ELEMENTS CONTINUED

`p:first-line` Target the first line of text

`p:first-letter` Target the first letter

The following pseudo-elements are not in the specification and currently have varying implementations in the different browsers. They also require the double colon pseudo-element syntax.

`p::-moz-selection`
`p::selection`

Style sections that have been highlighted by the user

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam in pharetra ligula, eget maximus leo. Aenean pretium mi et mauris mollis malesuada.

`input::-webkit-input-placeholder`
`input::-ms-input-placeholder`
`input::-moz-placeholder`
`input::placeholder`

Style an input element's placeholer text

These don't work when comma separated at the moment.

Placeholder Text

STATE BASED PSEUDO-CLASSES (The boring pseudo-classes)

`:link` Selects all unvisited links

`:hover` Selects elements on mouse hover

`:visited` Selects all visited links

`:active` Selects an element whilst it is being activated by the user, for example, when the user is mid-click

`:focus` Selects elements (typically form elements) that have been focused on via a click or keyboard event

FORM & VALIDATION PSEUDO-CLASSES (More mostly boring pseudo-classes)

`:default` Selects form elements that are in their default state

`:disabled` Selects form elements that are in a disabled state

`:enabled` Selects form elements that are not in a disabled state

`:in-range` Applies to elements that have range limitations; e.g. **<input type="number" min="0" max="5">**

4

`:out-of-range` A value of 4 would match :in-range
A value of 6 would match :out-of-range

6

Form & Validation pseudo-classes continued...

FORM & VALIDATION PSEUDO-CLASSES CONTINUED

`:valid` `:invalid`	Selects form elements whose contents are valid or invalid according to their type; e.g. **<input type="email">**
`:required`	Selects form elements that have the "required" HTML attribute
`:optional`	Selects form elements that **don't** have the "required" HTML attribute
`:read-write`	Selects elements that are user-editable, such as form input elements or elements that have the "contenteditable" HTML attribute
`:read-only`	Selects elements that are not user-editable (anything that doesn't match :read-write)
`:indeterminate`	Selects form elements that are in an indeterminate state; e.g. radio buttons can usually be checked or unchecked, but can sometimes be neither
`:checked`	Targets radio buttons, checkboxes and select menu <option> elements when they have been selected by the user. This can be particularly powerfuly, enabling what has come to be known as "**The Checkbox Hack**" (see page 6).

STRUCTURAL PSEUDO-CLASSES (The more fun pseudo-classes... but they get funner)

`:first-child`	Selects the first child, regardless of type	h1 p p ul p
`p:first-of-type`	Selects the first of a specific type of element	h1 p p ul p
`:last-child`	Selects the last child, regardless of type	h1 p p ul p
`p:last-of-type`	Selects the last of a specific type of element	h1 p p ul p
`:only-child`	Selects an element if it is the sole child and has no other siblings	ul p ul p
`p:only-of-type`	Selects a specific type of element if it is the sole child and has no other siblings	ul p ul p
`:nth-child(2)`	Selects the 2nd child, regardless of type	h1 p p ul p
`p:nth-of-type(2)`	Selects the 2nd of a specific type of element	h1 p p ul p
`:nth-last-child(2)`	Selects the 2nd child, regardless of type, but counting from the end	h1 p p ul p
`p:nth-last-of-type(2)`	Selects the 2nd of a specific type of element, but counting from the end	h1 p p ul p

MISCELLANEOUS PSEUDO-CLASSES

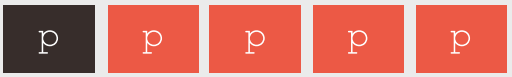
`:root` Selects the highest parent element in a document, typically the html element

`:lang(en)` Selects an element with the “lang” HTML attribute applied

`:empty` Selects an element that is **completely** empty

```
<div></div> /* Empty */
<div> </div> /* Not empty */
```

`p:not(:first-child)` **The Negation Pseudo-class** - Targets elements except for a specified variant of that element



`:target` Selects an element whose id value is currently being targeted via a # in the URL

```
<div id="Lorem">A</div>

div { color: black; }
div:target { color: red; }
```

url.com **A**

url.com#Lorem **A**

NTH-CHILD EXPRESSIONS (The fun stuff)

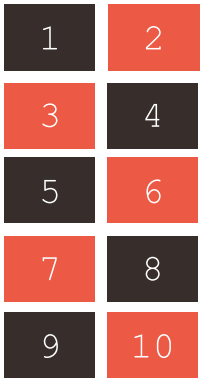
`:nth-child(2n+1)` **Target an infinite sequence**

Best way to memorise this is:
The first number is the sequence (every 2 elements)
The second number is where the sequence starts (the 1st element)

`:nth-child(-n+3)` Using negative numbers to select the **first** x amount of items

`:nth-last-child(-n+3)` Using negative numbers to select the **last** x amount of items

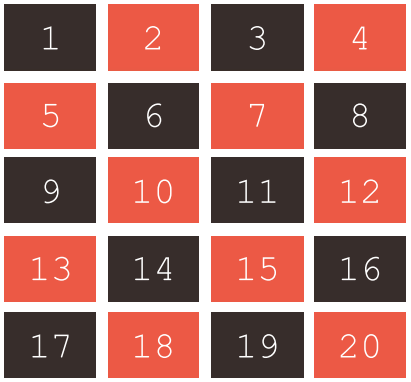
`:nth-child(n+2):nth-child(-n+4)` Combining nth-child expressions to **select an isolated range of items**



Using multiple selectors for more abstract sequences

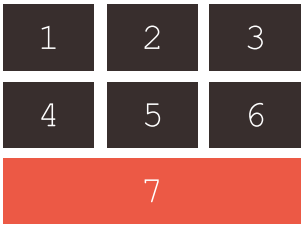
`:nth-child(4n+2), :nth-child(4n+3)`

`:nth-child(8n+2), :nth-child(8n+4), :nth-child(8n+5), :nth-child(8n+7)`



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

Target the last item if total items = x
This example targets the last child only if there are 7 items in total

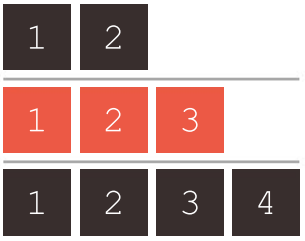



```
li:nth-child(4):nth-last-child(4),
li:nth-child(4):nth-last-child(4) ~ li {
  width: 25%;
}
```

Target the last *few* items if total items = x
This example targets the last 4 items but only if there are 7 items in total

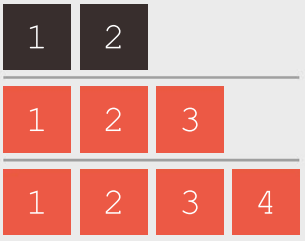

```
li:first-child:nth-last-child(3),
li:first-child:nth-last-child(3) ~ li {
  background: orange;
}
```

Target *all* items if total items = x
This example targets the items if there are exactly 3 items in total



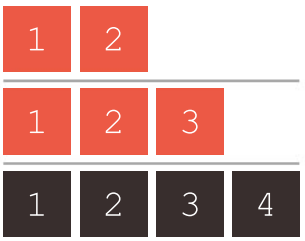

```
li:first-child:nth-last-child(n+3),
li:first-child:nth-last-child(n+3) ~ li {
  background: orange;
}
```

Target all items if total items = x *or greater*
This example targets the items if there are 3 items or greater in total



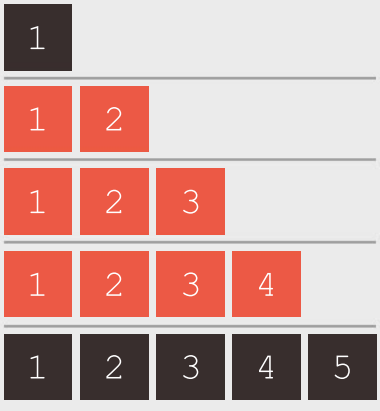

```
li:first-child:nth-last-child(-n+3),
li:first-child:nth-last-child(-n+3) ~ li {
  background: orange;
}
```

Target all items if total items = x *or fewer*
This example targets the items if there are 3 items or fewer in total




```
li:first-child:nth-last-child(n+2):nth-last-child(-n+4),
li:first-child:nth-last-child(n+2):nth-last-child(-n+4) ~ li {
  background: orange;
}
```

Target all items if total items is *between x and y*
This example targets the items if there are between 2 and 4 items in total



These selectors will work in IE9+. Kudos to Heydon Pickering and Lea Verou for making me aware of these techniques.

The Checkbox Hack - In its simplest form, it can enable easy custom form controls

```
<input type="checkbox" id="abc">
<label for="abc">Option 1</label>
```

```
input {
  opacity: 0;
  position: absolute;
}

input + label {
  background: black;
}

input:checked + label {
  background: orange;
}
```

Option 1
Option 2
Option 3 ✔
Option 4

When the input and label have corresponsing “id” and “for” attributes, the label becomes clickable on behalf of the checkbox input.

We can then hide the actual checkbox input and style the label however we want.

We can use the :checked pseudo-class and the adjacent sibling combinator (+) to style the currently selected option however we want.

The “Lobotomised Owl Selector” - Made famous by Heydon Pickering

```
* + * {
  margin-top: 1.5em;
}
```

The lobotomised owl selector targets **anything *that follows* anything**.

```
.sidebar > * + * {
  margin-top: 1.5em;
}
```

By combining with more combinators, its most practical use is to apply consistent vertical margins to specific sections of your layout.

This example would target **anything that follows anything and is a direct child of the .sidebar element**, and then apply a margin-top of 1.5em.

Using Pseudo-elements to Output Attribute Values

```
p:after {
  content: attr(datetime);
}
```

Pseudo-elements are all the rage, but less mainstream is their ability to output the value of a HTML attribute on the selected element.

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

This example would output an <a> element’s href value in brackets after the link; which could be particularly useful for print stylesheets.