Native CSS nesting

Note:

This feature is under development and there are currently no implementations in browsers, even behind flags. **Do not use this in production unless you transpile with PostCSS or SASS**.

One of the things that drove me to SASS was the ability to nest selectors to make the code easier to reason through and understand. For the longest time I've used something like this to style my links:

```
a {
  color: $color__link;
  text-decoration: none;
  text-decoration-skip-ink: auto;

  &:visited {
    color: $color__link-visited;
    text-decoration: underline;
    text-decoration-skip-ink: auto;
  }

  &:hover,
  &:focus,
  &:active {
    color: $color__link-hover;
    text-decoration: underline;
    text-decoration-skip-ink: auto;
  }

  &:focus {
    outline: thin dotted;
  }
}
```

Seeing the nested command makes it easier to see that they are related and the & is the character that the SASS developers chose to indicate the relationship between the parent selector and the child. The code will produce the following CSS declarations.

```
a {
  color: #000;
  text-decoration: none;
  text-decoration-skip-ink: auto;
}

a:visited {
  color: #F00D;
  text-decoration: underline;
  text-decoration-skip-ink: auto;
}

a:hover,
a:focus,
a:active {
  color: #F00D;
  text-decoration: underline;
  text-decoration-skip-ink: auto;
}

a:focus {
  outline: thin dotted;
}
```

Native CSS nesting provides the & selector and also a `@nest` at-rule that provides an explicit way to nest selectors.

The post will look at both the selector and the at-rule and explain any differences that may be there.

The & selector

The & selector allows you to nest rules together. This is similar to, but not quite the

same as, the way that SASS handles nesting.

We can use & on their own to indicate that we're nesting rules.

```
.foo {
  color: blue;
  & > .bar { color: red; }
}
/* equivalent to
  .foo { color: blue; }
  .foo > .bar { color: red; }
*/
```

We can also use it in a compound selector, refining the parent's selector.

```
.foo {
  color: blue;
  &.bar { color: red; }
}
/* equivalent to
  .foo { color: blue; }
  .foo.bar { color: red; }
*/
```

You can also use & in multiple selectors list but all the selectors on the list must all start with &

```
.foo, .bar {
  color: blue;
  & + .baz, &.qux { color: red; }
}
/* equivalent to
  .foo, .bar { color: blue; }
  :is(.foo, .bar) + .baz,
  :is(.foo, .bar).qux { color: red; }
*/
```

The parent and/or selector can be arbitrarily complicated. Note that the possible equivalencies use the :is() pseudo-class to provide a more compact equivalent.

```
.error, #404 {
  &:hover > .baz {
    color: red;
  }
}
/* equivalent to
  :is(.error, #404):hover > .baz {
    color: red;
  }
*/

.foo {
  &:is(.bar, &.baz) {
    color: red;
  }
}
/* equivalent to
  .foo:is(.bar, .foo.baz) {
    color: red;
  }
*/
```

Multiple levels of nesting “stack up” the selectors. There is no real limit to how deep you can nest selectors, just make sure you don’t dig so deep that the chains become impossible to read.

```
figure {
  margin: 0;

  & > figcaption {
    background: hsl(0 0% 0% / 50%);

    & > p {
      font-size: .9rem;
```

```
    }
  }
}
/* equivalent to
  figure { margin: 0; }
  figure > figcaption { background: hsl(0 0% 0% / 50%); }
  figure > figcaption > p { font-size: .9rem; }
*/
```

The @nest at-rule

The `@nest` rule functions identically to a nested style rule: it starts with a selector, and contains a block of declarations that apply to the elements the selector matches. That block is treated identically to a style rule's block, so anything valid in a style rule (such as additional @nest rules) is also valid here.

The only difference between @nest and rule using & is that the selector used in a @nest rule is less constrained:

- It only must contains a nesting selector in it somewhere, rather than requiring it to be at the start of each selector
- A list of selectors is nest-containing if all of its individual complex selectors are nest-containing

```
.foo {
  color: red;
  @nest & > .bar {
    color: blue;
  }
}
/* equivalent to
  .foo { color: red; }
  .foo > .bar { color: blue; }
*/
```

You can nest `@nest` rules, just like you do with regular nesting selectors.

```
figure {
  margin: 0;

  @nest & > figcaption {
    background: hsl(0 0% 0% / 50%);

    @nest & > p {
      font-size: .9rem;
    }
  }
}
/* equivalent to
  figure { margin: 0; }
  figure > figcaption { background: hsl(0 0% 0% / 50%); }
  figure > figcaption > p { font-size: .9rem; }
*/
```

`@nest` allows selectors that don't start with an &, so the following are also valid:

```
.foo {
  color: red;
  @nest .parent & { color: blue; }
}
/* equivalent to
  .foo { color: red; }
  .parent .foo { color: blue; }
*/

.foo {
  color: red;
  @nest :not(&) {
    color: blue;
  }
}
/* equivalent to
  .foo { color: red; }
  :not(.foo) { color: blue; }
```

```
*/
```

You can also combine the two types of nesting in the same rule

```
.foo {
  color: blue;
  @nest .bar & {
    color: red;
    &.baz {
      color: green;
    }
  }
}
/* equivalent to
  .foo { color: blue; }
  .bar .foo { color: red; }
  .bar .foo.baz { color: green; }
```

Browser support and resources

As mentioned in the note at the top of the post, there is no support for CSS nesting in any browser yet.

> Check Caniuse for more information.

Additional resources

- Specification — drafts.csswg.org
- Chrome support bug — bugs.chromium.org
- Safari support bug — bugs.webkit.org
- Firefox support bug — bugzilla.mozilla.org