CSS Media Query Range Syntax

We rely on CSS Media Queries for selecting and styling elements based on a targeted condition. That condition can be all kinds of things but typically fall into two camps:

(1) the type of media that's being used, and (2) a specific feature of the browser, device, or even the user's environment.

So, say we want to apply certain CSS styling to a printed document:

```
@media print {
    .element {
        /* Style away! */
    }
}
```

The fact that we can apply styles at a certain viewport width has made CSS Media Queries a core ingredient of responsive web design since Ethan Marcotte coined the term.
If the browser's viewport width is a certain size, then apply a set of style rules, which allows us to design elements that respond to the size of the browser.

```
/* When the viewport width is at least 30em... */
@media screen and (min-width: 30em) {
    .element {
        /* Style away! */
    }
}
```

Notice the `and` in there? That's an operator that allows us to combine statements. In that example, we combined a condition that the media type is a `screen` and that it's `min-width` feature is set to `30em` (or above).
We can do the same thing to target a range of viewport sizes:

```
/* When the viewport width is between 30em - 80em */
@media screen and (min-width: 30em) and (max-width: 80em) {
  .element {
    /* Style away! */
  }
}
```

Now those styles apply to an explicit range of viewport widths rather than a single width!
But the Media Queries Level 4 specification has introduced a new syntax for targeting a range of viewport widths using common mathematical comparison operators — things like `<`, `>`, and `=` — that make more sense syntactically while writing less code.

Let's dig into how that works.

New comparison operators

That last example is a good illustration of how we've sort of "faked" ranges by combining conditions using the `and` operator.

The big change in the Media Queries Level 4 specification is that we have new operators that compare values rather than combining them:

- `<` evaluates if a value is **less than** another value
- `>` evaluates if a value is **greater than** another value
- `=` evaluates if a value is **equal** to another value

- `<=` evaluates if a value is **less than or equal t**o another value
- `>=` evaluates if a value is **greater than or equal t**o another value

Here's how we might've written a media query that applies styles if the browser is `600px` wide or greater:

```
@media (min-width: 600px) {
  .element {
    /* Style away! */
  }
}
```

Here's how it looks to write the same thing using a comparison operator:

```
@media (width >=600px) {
  .element {
    /* Style away! */
  }
}
```

Targeting a range of viewport widths

Often when we write CSS Media Queries, we're creating what's called a breakpoint — a condition where the design "breaks" and a set of styles are applied to fix it.
A design can have a bunch of breakpoints! And they're usually based on the viewport being between two widths: where the breakpoint starts and where the breakpoint ends.
Here's how we've done that using the `and` operator to combine the two breakpoint values:

```
/* When the browser is between 400px - 1000px */
@media (min-width: 400px) and (max-width: 1000px) {
    /* etc. */
}
```

You start to get a good sense of how much shorter and easier it is to write a media query when we ditch the Boolean `and` operator in favour of the new range comparison syntax:

```
@media (400px <=width <=1000px) {
    /* etc. */
}
```

Much easier, right? And it's clear exactly what this media query is doing.

KEEP CODING. ..KEEP LEARNING

Do Follow on Instagram
@SKY.GIT