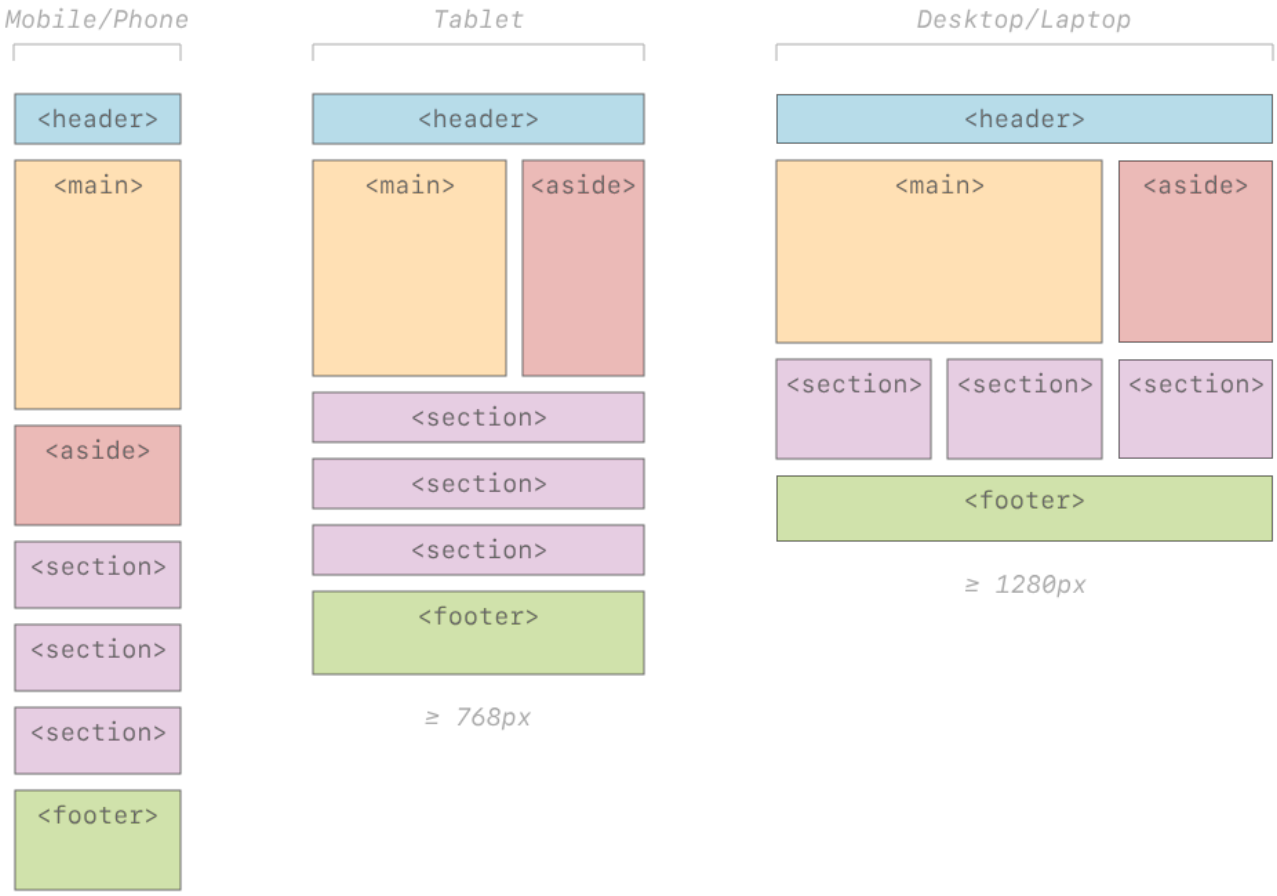
Responsive Design

Unlike print, digital design is experienced on a variety of formats (for example, phones, tablets, a variable browser window), so as designer's we need to create a system that works on many different formats. As developers, we need to keep this in mind for the same reasons.

"This is just the beginning" - Brad Frost in 2012



A typical/example responsive layout, adjusting the layout to reflow based on the device width.
credit: michael fehrenbach

The Viewport

You've seen this before in the `<head> </head>` section.

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

(Start adding it if you haven't already!)

The `width=device-width` tells the browser to use whatever the screen's actual pixel dimension is, and `initial-scale=1` sets the starting zoom for the page to 100%.

This is **how the browser knows how to make the page respond,** and how our CSS rules know what width to use.
We call the portion of the page visible at one time the viewport.

https://youtu.be/VQKMoT-6XSg

https://youtu.be/VQKMoT-6XSg?t=2474s

Using Media Queries

Media queries are a CSS technique that allow you to target specific capabilities of a device. It uses the `@media` rule to define how the CSS will change only if that condition is true.

They allow us to check if screen is a certain width or resolution (or other features, which we'll get to)—and then apply selective CSS only in that scenario.

Practically, these are blocks of CSS—a little bit like *selectors* that contain other selectors—but which only apply conditionally when the test/criteria is met.

```
/* If the browser window is smaller than 650px, the background color will change to black and the text to white: */
@media (max-width: 650px) {
  body {
    background-color: black;
    color:  white;
  }
}
```

Media queries can be used to target vertical rules as well:

```
/* If the browser window is shorter than 650px, the background
will turn orange */

@media (max-height: 650px) {
  body {
    background-color: orange;
  }
```

Specifying a `max-width` or `max-height` in a media query is often used when you write CSS for your website viewed on desktop, but if you want it to look different at a smaller, or larger, breakpoint. It's also helpful in fine-tuning some of your design elements on a variety of sizes.

A tip on how to differentiate between min & max:
`min-width`: Adds styles for larger screens (expanding layout as screen grows).
`max-width`: Adds styles for smaller screens (shrinking layout as screen gets smaller).

Width will vary the most between devices—from the `375px`–`428px` of your phones, through the ~ `1440px`–`1680px` of your laptops, up to the ~ `2560px`–`3440px` you might see on desktop displays.

Since this `width` is usually our primary design constraint (`height` being handled with scrolling), we need *width-based* media queries to adjust our layouts across this wide range, lest our designs "break."

It can be helpful to think of media queries as conditional *if* statements. "If this thing is true, then do this." (You may have heard of *If This Then That*.) We'll talk about this later with JavaScript, where conditionals are ubiquitous and powerful.

The points at which media queries take affect are also known as breakpoints.

Here are some typical breakpoints

`< 768px` Phones
`≥ 768px` Tablets
`≥ 992px` Medium desktops
`≥ 1200px` Large desktops

```
/* The body's font-size is smaller by default */
body {
  font-size: 1em;
}

/* Tablet breakpoint */
@media (min-width: 768px) {
  body {
    font-size: 3em;
  }
}

/* Desktop breakpoint */
@media (min-width: 992px) {
  body {
    font-size: 4em;
  }
}

/* Wide screen breakpoint */
@media (min-width: 1200px) {
  body {
    font-size: 5em;
  }
}
```

A Very Practical Use Case: Typography with Variables

```
:root {
        --font-size: 16px;
        --spacing: 20px;
}

@media (min-width: 400px) {
        :root {
                --font-size: 24px; /* These numbers go up for
larger screens */
                --spacing: 40px;
        }
}
```

https://codepen.io/Divya-Mehra/embed/EaxNdPB

https://codepen.io/Divya-Mehra/pen/EaxNdPB

Mobile-first design

So this can all get very complicated, very quickly—especially with complex designs, overlapping rules, and the wide ranges of devices to consider. One of the easiest methodologies to keep things understandable is practicing *mobile first* design/development. This has become kind of *buzzwordy* in the past decade or so, but it is a good philosophy to adhere to nonetheless.

Your design constraints will be tighter, by tackling your smallest layout first—but it is almost always easier to scale things up than down. A mobile design can always work as a passable desktop one; the reverse is rarely true.

(Another way to think of it: *if it doesn't work on mobile, it doesn't work.*)

- In your design, *mobile first* means considering small screens and *then* adding complexity, limits, or considerations for larger/desktop screens.

- In your HTML/CSS development, this means writing your styles for mobile... first, *then* adding `min-width` breakpoints (below them) to progressively enhance your design as it scales up.

In case you're interested: Some thoughts & tips on container queries from Josh W Comeau

The Beginnings of a Mobile Nav

https://codepen.io/Divya-Mehra/embed/ogNYaZx

https://codepen.io/Divya-Mehra/pen/ogNYaZx