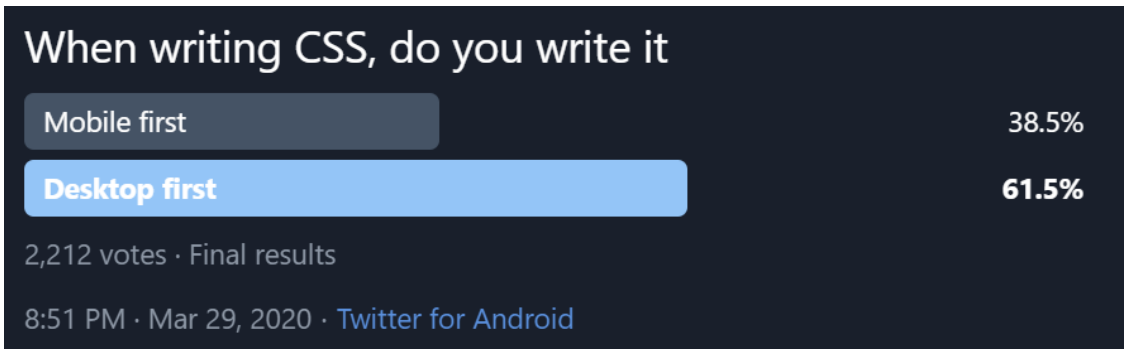
APRIL 7, 2020 / #RESPONSIVE DESIGN

How to Take the Right Approach to Responsive Web Design

Kevin Powell

I ran a poll on Twitter awhile ago, and the results surprised me.

Not only did I expect the results to be the other way around, **I thought that mobile-first would get at least 80% of the vote.**



Desktop-first wins with more than 61% of the vote!

In the replies, some people explained why they write desktop-first. The general themes of those reasons:

- They learned CSS writing it Desktop only, so this seemed like the natural progression
- Clients want to see the desktop version

What is mobile-first

Mobile-first is when we start by **writing our CSS for mobile devices and then use media queries to add in styling for larger screen sizes**.

In general, that means that media queries use a `min-width`. We're using media queries to add or overwrite styles for a set breakpoint and bigger, such as this example:

```
.sales-points {
  padding: 3em 0;
}

@media (min-width: 600px) {
  .sales-points {
    display: flex;
    justify-content: space-between;
  }
}
```

In that example, for small screens we're simply applying some padding. Assuming this section of the site has children in it, we turn those children into columns at a minimum width of `600px`.

So when the viewport is `600px` or larger, we will have columns. The rest of the time, things stack.

As you've probably guessed, **a desktop-first approach is the other**

Why mobile-first is easier

Websites are naturally responsive before we even write a single line of CSS.

If you remove the CSS from any page on the internet, even some site made in for a very specific screen size back in 2001, you now have a responsive, mobile-friendly website!

Desktop styles tend to be more complex

When we style for desktop-first, we're adding widths, columns, and moving things around. We're adding complexity. We're doing this for good reason, as we have more real-estate to work with.

Not only do we want to take advantage of that to make things look more interesting, but **if we didn't make things more complex on larger screens, things wouldn't look very good**. Even if you have a *very* simple website, you don't want it to have text stretching from one side to the other.

Look at what an article here on FCC News would look like if the text went from one side to the other.

We can all agree that you'd never read something like that, yes? I literally have to move my head a little from left to right to read a full line on my screen. It's terrible.

Mobile layouts tend to be very simple, making it very easy to start there

For all the people who replied to me saying that their clients preferred seeing the desktop version, or that they were only given desktop comps by their designers, I would argue it's still easier to start mobile-first.

For many sites, once you've set up your typography, you're 70% of the way there. Things like:

- `font-family`
- `font-size`
- `font-weight`
- `margin` (on your text elements)

Next up, you can go and do some very basic layout styling on your layout elements, such as:

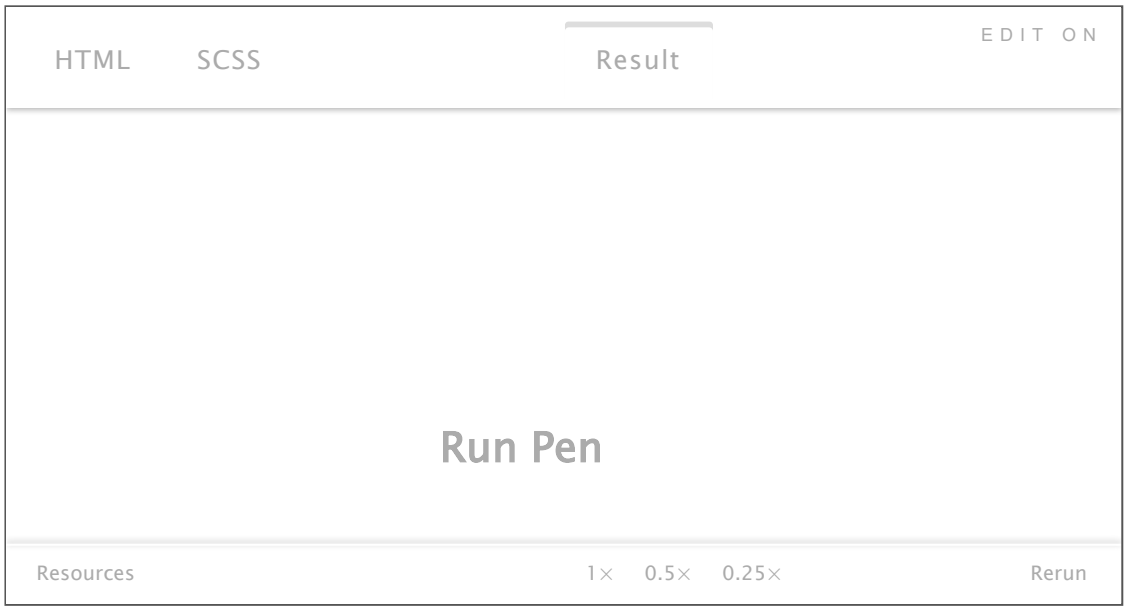
- `padding`
- `background-color`
- `color`
- and maybe some tweaks with `margin`

If you were feeling particularly lazy, or have a very simple site, you could stick a `max-width` on your container and be done with the entire thing and not even have to worry about a media query at all!

Most of the time, we do want to up the game at bigger screen sizes though, and that's why I feel like mobile-first is the way to go. It's the natural progression upward.

Comparing mobile-first to desktop-first

Below is a CodePen that has a *very* simple layout put together using a desktop-first and mobile-first approach.



If you open the pen up and play around with the viewport size, you'll see that the end result is exactly the same.

But if the end result using either approach is exactly the same, why

Desktop-first can lead to redundant code

In the above pen, the desktop-first approach uses the following code:

```
/*  desktop-first */
.desktop-first .sales-points {
  display: flex;
  justify-content: space-between;
}

.desktop-first .sales-point {
  width: 30%;
}

@media (max-width: 600px) {
  .desktop-first .sales-points {
    display: block;
  }

  .desktop-first .sales-point {
    width: 100%;
  }
}
```

As you can see in the CodePen, it works perfectly fine, but there is a bunch of code in here that's made redundant when we use a desktop-first approach.

Notice how we first declare a `display: flex` only to put it back to the default `display: block` in the media query. Also, for our columns, we change the `width` and then, once again, go back to the default later on.

The mobile-first approach has a lot less redundant code. Because there was no styling of the text or background colors, there is no

```
/*  mobile-first */
@media (min-width: 600px) {
  .mobile-first .sales-points {
    display: flex;
    justify-content: space-between;
  }

  .mobile-first .sales-point {
    width: 30%;
  }
}
```

Going back to the defaults should be a red flag

I realize that some things are more complex than this (and we'll be getting there soon), but most of what I'm worrying about here is from a layout perspective.

For the layout I created above, I didn't write one line of code for the mobile-first approach. I just relied on how the document was flowing from the start. In the desktop-first approach, I have to tackle both because I need to reset things back to their default state.

The fact that I'm resetting things like `display` and `width` to their default state, to me, is a red flag. It means I'm writing something that could have been avoided. That means I'm wasting my time.

Some things aren't so simple

Some components look completely different at different screen sizes, such as navigation menus. Other times, you have **styles on**

In the below video, I run into that exact issue where I needed to move an element using `position: absolute` for smaller screens. Rather than have to position it, then reset the position back to the default at larger screen sizes, it seemed like a logical choice for a `max-width` media query.

If you hit play on the video, it should start right where I tackle this issue if you'd like to see it in action (17:41 just in case it doesn't start at the right spot).

So sometimes there are exceptions, and there is nothing wrong with that. My point here isn't that we should be robots who do things one way. There are times when different approaches make sense, but **I do like to believe having a general rule of thumb helps**.

So next time you are designing a site, **even if you only have a desktop mock-up to go by, try starting mobile first**. It doesn't take any more work at all, and in the long run I bet it'll save you a ton of redundant code. It's pretty simple too!

2. Add in colors and padding
3. Put anything layout related into a `min-width` media query

When you're done with your layout, not only will you have knocked out that desktop version that your client is dying to see, but you'll be 90% of the way there in your mobile one as well, without having even really thought about it.

Do you struggle with making things responsive?

Making websites responsive is a topic that a lot of people tell me they struggle with. To help, **I've created a free course called Conquering Responsive Layouts**. It's put together as a 21-day challenge in which we'll cover a topic a week, with each one adding onto what we already learned.

I realize that we're all busy with kids, family, work, and more, so each day will only be 10-30 minutes worth of lessons, with 2-3 lessons a week. In between you'll have small challenges to complete, working your way up to being comfortable making responsive layouts.

The course is launching on the 13th of April and because it's a 21-day course, the doors close on that day. Click here to sign up to start conquering responsive layouts!

If you're reading this after the fact, you can go and sign up for the next time it launches, but it won't open up again for a few months.

If you read this far, tweet to the author to show them you care.

Tweet a thanks

Learn to code for free. freeCodeCamp's open source curriculum has helped more than 40,000 people get jobs as developers.

Get started

freeCodeCamp is a donor-supported tax-exempt 501(c)(3) nonprofit organization (United States Federal Tax Identification Number: 82-0779546)

Our mission: to help people learn to code for free. We accomplish this by creating thousands of videos, articles, and interactive coding lessons - all freely available to the public. We also have thousands of freeCodeCamp study groups around the world.

Donations to freeCodeCamp go toward our education initiatives, and help pay for servers, services, and staff.

You can make a tax-deductible donation here.

Trending Guides

JavaScript split()

HTML Bullet Points

What is UX Design?

Dark Mode on Google

Contraction Grammar

What is a JSON file?

Python String Format

Span HTML

SQL Count

HTML Comment

Python strip()

HTML Select Tag

Insert into SQL

MVC Architecture

Check GPU in Windows
HTML File Text Editor
Responsive Web Design
Online Coding Classes
Python String to Array
Lambda Function Python
SCP Linux Command
SQL Distinct Statement
HEIC to JPG on Windows
Insert Checkbox in Word
Drop Pin on Google Maps
Rotate Screen Windows 10

Our Nonprofit