Responsive Images

In this lab, we will be covering the following:

- How to progressively enhance modern image formats
- Understanding the responsive images problem space
- Using `srcset` for resolution switching
- How to achieve art direction with `picture`
- Practical considerations for responsive images

Let's begin by looking at image formats and how different formats alone can significantly reduce the amount of data our website needs to send down the wire.

Using the picture element to progressively enhance images

We will soon see how to use the `picture` element for "art direction," serving up different images for different situations. However, I have actually found the greatest practical utility in the `picture` element is for providing alternative image formats. Let's look at that first.

The most common use case I find myself currently needing is providing an AVIF version of an image, if possible, then a WebP version for users if they can't make use of AVIF, and then a JPG if a user can't make use of an AVIF or a WebP. Here is how we can do that with `picture`:

```
<picture>
    <source srcset="scone.avif" type="image/avif" />
    <source srcset="scone.webp" type="image/webp" />
    <img
        src="scone.jpg"
        alt="A delicious scone, baked to perfection"
        loading="lazy"
    />
</picture>
```

The essence of the `picture` element is wrapping a number of images up, with the preferred options first, and the browser chooses the first one it is able to make use of. If you have already read *Lab 2* on *Writing HTML Markup*, you will recognize the `source` element from the *Providing alternate media sources* section. The principle is identical: we specify different versions of a piece of media, and the browser chooses the first one it can make use of.

In our example code above, we would prefer the browser to use the `scone.avif` image, but if it can't use that, it uses the `scone.webp`. If even that cannot be understood, it falls back to `scone.jpg`. Crucially, in this situation where all the images are of the same thing, you don't need to write the `alt` text for the image more than once. Whichever image the browser chooses, it will only download that one image.

In addition, back in *Lab 2*, we covered the `loading` attribute, and noted that if we were happy for images to download in their own time, we could add `loading="lazy"` to them. The same is true here, but like the `alt` tag, you can see in our example code that we only have to specify it on the `img` tag; the two `source` elements in our `picture` are merely alternate `src` values for the `img`.

So that is how we can provide modern images for browsers that support them, and older, more broadly supported formats to those that don't, all the while providing solid accessibility and meaning with the `alt` tag in case the image doesn't load or the user can't make use of the visuals.

In the prior code, we are just dealing with a single image and different formats of it. That addresses our problem of file weight. Now let's look at how to deal with serving the best image given the situation. For that, we will put aside `picture` momentarily and make use of another tool, `srcset`. But first, let's be certain we understand the problem we are trying to solve.

Simple resolution switching with srcset

Let's suppose you have three versions of the same image. One is suitable for standard pixel density displays, commonly referred to as "1x". Another image is higher resolution, to cater for higher density, or "1.5x" displays, and, finally, the third is a high "2x" resolution version for very high-density displays. Here is how we can let the browser know that we have these three versions available:

```
<img
  src="scones_small.jpg"
  srcset="scones_medium.jpg 1.5x, scones_large.jpg 2x"
  alt="a delicious looking baked scone"
/>
```

This is about as simple as things get with responsive images, so let's ensure that the syntax makes perfect sense.

First of all, the `src` attribute, which you will already be familiar with, has a dual role here; it's specifying the small 1x version of the image, and it also acts as a fallback image if the browser doesn't support the `srcset` attribute. That's why we are using it for the small image. This way, older browsers that will ignore the `srcset` information will get the smallest and best-performing image possible.

For browsers that understand `srcset`, with that attribute, we provide a comma-separated list of images that the browser can choose from. After the image name (such as `scones_medium.jpg`), we issue a simple hint.

I've specifically called it a hint rather than an instruction or a command, and you will see why in a moment. In this example `1.5x` and `2x` have been used, but any integer would be valid. For example, `3x` or `4x` would work too (providing you can find a suitably high-resolution screen).

However, there is an issue here; a device with a `1440px` wide, `1x` screen will get the same image as a `480px` wide, `3x` screen. That may or may not be the desired effect.

Advanced switching with srcset and sizes

Let's consider another situation. In a responsive web design, it wouldn't be uncommon for an image to be the full viewport width on smaller viewports, but only half the width of the viewport at larger sizes. The main example in *Lab 1*, *The Essentials of Responsive Web Design*, was a typical illustration of this. Here's how we can communicate these intentions to the browser:

```
<img
  srcset="scones-small.jpg 450w, scones-medium.jpg 900w"
  sizes="(min-width: 280px) 100vw, (min-width: 640px) 50vw"
  src="scones-small.jpg"
  alt="Lots of delicious scones"
/>
```

Inside the `img` tag, we are utilizing `srcset` again. However, this time, after specifying the images, we are adding a value with a `w` suffix. This tells the browser how wide the image is. In our example, we have a 450 px wide image (called `scones-small.jpg`) and a 900 px wide image (called `scones-medium.jpg`). It's important to note that

this `w`-suffixed value isn't a "real" size. It's merely an indication to the browser, roughly equivalent to the width in "CSS pixels."

This `w`-suffixed value makes more sense when we factor in the additional `sizes` attribute. The `sizes` attribute allows us to communicate the intentions for our images to the browser. In the example above, the first value is equivalent to "for devices that are at least `280px` wide, I intend the image to be around `100vw` wide."

If some of the units used, such as `vh` (where `1vh` is equal to 1% of the viewport height) and `vw` (where `1vw` is equal to 1% of the viewport width), don't make sense, be sure to read *Lab 6, CSS Selectors, Typography, and More*.

The second part is, effectively, "Hi browser, for devices that are at least `640px` wide, I only intend the image to be shown at `50vw`." That may seem a little redundant until you factor in **dots per inch** (**DPI**) (or **DPR** for **device pixel ratio**). For example, on a `320px` wide device with a `2x` resolution (effectively requiring a `640px` wide image, if shown at full width), the browser might decide the `900px` wide image is actually a better match as it's the first option it has for an image that would be big enough to fulfill the required size.

Art direction with the picture element

The final scenario you may find yourself in is one in which you have different images that are applicable at different viewport sizes. For example, consider our cake-based example again from *Lab 1*. Maybe on the smallest screens we would like a close-up of the scone with a generous helping of jam and cream on top. For larger screens, perhaps we have a wider image we would like to use. Perhaps it's a wide shot of a table loaded up with all manner of cakes. Finally, for larger viewports still, perhaps we want to see the exterior of a cake shop on a village street with people sitting outside eating cakes and drinking tea (I know, sounds like nirvana, right?).

We need three different images that are most appropriate at different viewport ranges. Here is how we could solve this with `picture`:

```
<picture>
  <source media="(min-width: 480px)" srcset="cake-table.jpg" />
  <source media="(min-width: 960px)" srcset="cake-shop.jpg" />
  <img src="scones.jpg" alt="Lots of cakes" />
</picture>
```

Be aware that when you use the `picture` element, it is merely a wrapper to facilitate other images making their way to the `img` tag within. If you want to style the images in any way, it's the `img` tag that should get your attention. Similarly, you can't change the `alt` tag contents by depending on which image gets loaded, so make sure you provide text that covers all eventualities as best you can.

Secondly, the `srcset` attribute here works exactly the same as in the previous example.

Thirdly, the `img` tag provides your fallback image and also the image that will be displayed if a browser understands `picture` but none of the media definitions match.

Just to be crystal clear, do not omit the `img` tag from within a `picture` element or things won't end well!

The key difference with `picture` is that we have a `source` tag. Here, we can use media query-style expressions to explicitly tell the browser which asset to use in a matching situation. Our first one in the preceding example is telling the browser, "Hey you, if the screen is at least 480 px wide, load in the `cake-table.jpg` image instead." As long as the conditions match, the browser will dutifully obey.

Summary

In this lab, we looked at new image formats like AVIF and WebP, and the massive economies they can offer in terms of file size. We considered their shortcomings in terms of support and looked at how we can use the `picture` element to serve up both those image types alongside a suitable fallback.

We also looked at `srcset` and how that allows us to provide the browser with a number of possibilities alongside our preference, enabling the browser to make an informed choice about which to use.