This is good example of the separation of content from presentation. A navigation menu *is* an unordered list, but it also makes sense to display them as buttons instead of a typical bulleted list. Intelligently designed HTML allows search engines to infer the structure of our content, while CSS lets us display it to humans in beautiful ways.

You can even create custom bullets for `<li>` elements with the `list-style-image` property (see MDN for details).

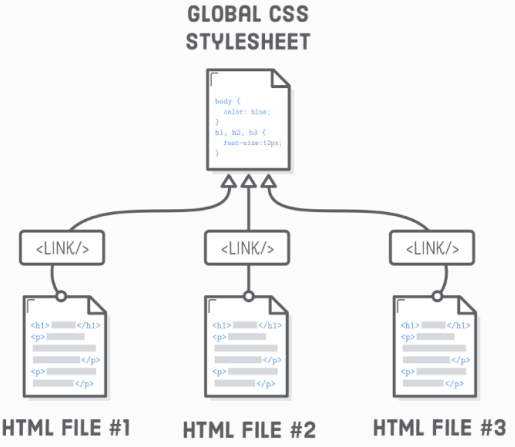
Defining the color of your text and the appearance of your bullets might seem trivial, and it kind of is. But, look at the bigger picture: this is about gaining *complete* control over the appearance of an HTML document. Alone, a single CSS property is silly. Put them all together, and you're able to create a totally customized web page.

REUSABLE STYLESHEETS

So, we just defined some basic styles for one of our web pages. It would be really convenient if we could reuse them on our other page, too. For this, all we need to do is add the same `<link/>` element to any other pages we want to style. Try adding the following line to the `<head>` of `dummy.html`:

```
<link rel='stylesheet' href='styles.css'/>
```

Now, our `dummy.html` pages should match our `hello-css.html` styles. Whenever we change a style in `styles.css`, those changes will automatically be reflected in both of our web pages. This is how you get a consistent look and feel across an entire website.



You'll almost always have at least one stylesheet that's applied to the entire site. It's usually a good idea to use root-relative paths when linking global stylesheets to avoid problems in nested pages. For example, `some-folder/page.html` would need to use `../styles.css` to reference our `styles.css` file, and this can get real confusing real quick.

MORE TEXT STYLES

There's a whole bunch of different CSS properties that we'll be introducing over the course of this tutorial, but for now, let's finish up with some of the most common ways to format text.

UNDERLINES

The `text-decoration` property determines whether text is underlined or not. By setting it to `none`, we can remove the default underline from all of our links. We'll discuss link styles in-depth later on.

```
a {
  text-decoration: none;
}
```

The other common value for `text-decoration` is `line-through` to strike out "deleted" text. But, remember that *meaning* should always be conveyed through HTML—not CSS. It's better to use the `<ins>` and `<del>` elements instead of adding a `line-through` style to, say, an ordinary `<p>` element.

TEXT ALIGNMENT

The aptly named `text-align` property defines the alignment of the text in an HTML element.

```
p {
  text-align: left;
}
```

Other accepted values are `right`, `center`, or `justify`, but notice how it always aligns to the entire page:

This isn't what you want for most websites. We'll learn why this is the case in the next chapter when we start talking about CSS boxes.

FONT WEIGHT AND STYLES

The `font-weight` property defines the "boldness" of the text in an element, and the `font-style` property indicates whether it's italicized or not.

Let's say we don't want our headings to be bold. Update our heading font rule in `styles.css` to match the following:

```
h1, h2, h3, h4, h5, h6 {
  font-family: "Helvetica", "Arial", sans-serif;
  font-weight: normal;                          /* Add this */
}
```

These properties clearly demonstrate the separation of content (HTML) from presentation (CSS). The following rules swap the appearance of the `<em>` and `<strong>` elements:

```
/* (You probably shouldn't do this) */
em {
  font-weight: bold;
  font-style: normal;
}

strong {
```

```
  font-weight: normal;
  font-style: italic;
}
```

We don't suggest doing this for real websites though. Font weights and styles will, however, become a lot more important once we start playing with custom fonts in the Web Typography chapter.

THE CASCADE

The "cascading" part of CSS is due to the fact that rules cascade down from multiple sources. So far, we've only seen one place where CSS can be defined: external `.css` files. However, external stylesheets are just one of many places you can put your CSS code.

The CSS hierarchy for every web page looks like this:

- The browser's default stylesheet
- User-defined stylesheets
- External stylesheets (that's us)
- Page-specific styles (that's also us)
- Inline styles (that could be us, but it never should be)

This is ordered from least to most precedence, which means styles defined in each subsequent step *override* previous ones. For example, inline styles will always make the browser ignore its default styles. The next few sections focus on the last two options because that's what we have control over as web developers (in addition to the external styles we've already been working with).

We made an effort to get you started down the right path with external stylesheets. It's important to understand page-specific and inline styles because you'll most definitely encounter them in the wild, but external stylesheets are by far the best place to define the appearance of your website.

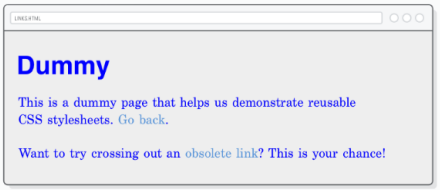
PAGE-SPECIFIC STYLES

The `<style>` element is used to add page-specific CSS rules to individual HTML documents. The `<style>` element always lives in the `<head>` of a web page, which makes sense because it's metadata, not actual content.

As an example, let's apply some styles to our `dummy.html` page by updating its `<head>` element to this:

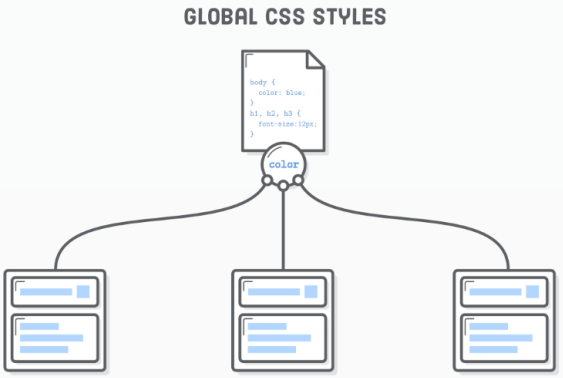
```
<head>
  <meta charset='UTF-8'/>
  <title>Dummy</title>
  <link rel='stylesheet' href='styles.css'/>
  <style>
    body {
      color: #0000FF;     /* Blue */
    }
  </style>
</head>
```

These apply *only* to `dummy.html`. Our `hello-css.html` page won't be affected. If you did it right, you should see bright blue text when you load `dummy.html` in a browser.



Anything you would put in our `styles.css` file can live in this `<style>` element. It uses the exact same CSS syntax as an external stylesheet, but everything here will override rules in our `styles.css` file. In this case, we're telling the browser to ignore the `color` property we defined for `<body>` in our external stylesheet and use `#0000FF` instead.

The problem with page-specific styles is that they're incredibly difficult to maintain. As shown in the above diagram, when you want to apply these styles to another page, you have to copy-and-paste them into *that* document's `<head>`. Trying to track down redundant CSS rules in multiple `.html` files is much harder than editing a single `.css` file.



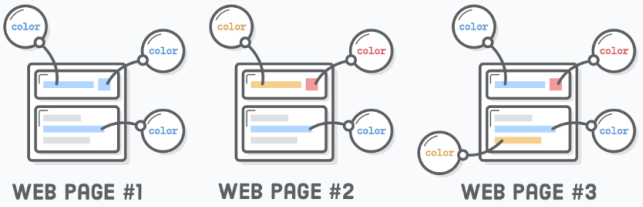
Page-specific styles occasionally come in handy when you're in a rush, but it's almost always better to store all your CSS in external stylesheets opposed to `<style>` elements.

INLINE STYLES

You can also stick CSS rules in the `style` attribute of an HTML element. In `dummy.html`, we have a link that doesn't actually go anywhere. Let's make it red via an inline style so we remember it's a dead link:

```
<p>Want to try crossing out an <a href='nowhere.html'
   style='color: #990000; text-decoration: line-through;'>obsolete link</a>?
   This is your chance!</p>
```

Like page-specific styles, this is the same CSS syntax we've been working with. However, since it's in an attribute, it needs to be condensed to a single line. Inline styles are the most specific way to define CSS. The `color` and `text-decoration` properties we defined here trump *everything*. Even if we went back and added a `text-decoration: none` to our `<style>` element, it wouldn't have any effect.



Inline styles should be avoided at all costs because they make it impossible to alter styles from an external stylesheet. If you ever wanted to re-style your website down the road, you can't just change a few rules in your global `styles.css` file—you'd have to go through every single page and update every single HTML element that has a `style` attribute. It's horrifying.

That said, there will be many times when you need to apply styles to only a specific HTML element. For this, you should always use CSS classes instead of inline styles. We'll explore classes in the CSS Selectors chapter.

MULTIPLE STYLESHEETS

CSS rules can be spread across several external stylesheets by adding multiple `<link/>` elements to the same page. A common use case is to separate out styles for different sections of your site. This lets you selectively apply consistent styles to distinct categories of web pages.

For instance, if we had a bunch of product pages that looked entirely different than our blog, we could use the following. (We don't actually have these stylesheets defined, so don't bother adding them to our example project.)

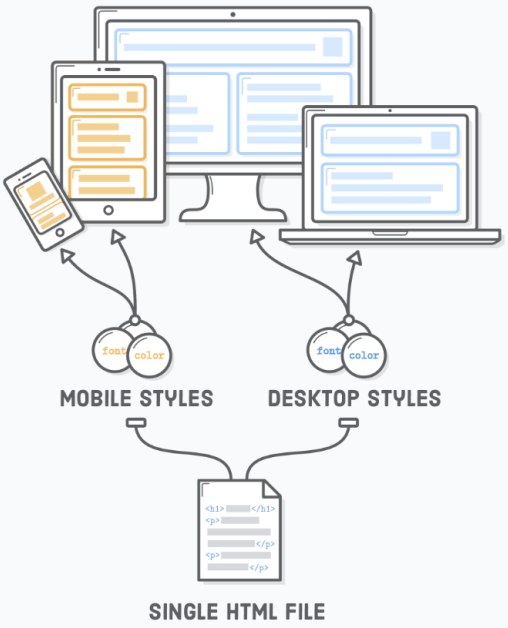
```
<!-- All product pages have this -->
<head>
  <link rel='stylesheet' href='styles.css'/>
  <link rel='stylesheet' href='product.css'/>
</head>
```

```
<!-- While all blog posts have this -->
<head>
  <link rel='stylesheet' href='styles.css'/>
  <link rel='stylesheet' href='blog.css'/>
</head>
```

The order of the `<link/>` elements matters. Stylesheets that come later will override styles in earlier ones. Typically, you'll put your "base" or "default" styles in a global stylesheet (`styles.css`) and supplement them with section-specific stylesheets (`product.css` and `blog.css`). This allows you to organize CSS rules into manageable files while avoiding the perils of page-specific and inline styles.

SUMMARY

We talked a lot about separating content from presentation in this chapter. This not only allows us to reuse the same CSS stylesheet in multiple HTML documents, but also lets us conditionally apply *different* CSS rules to the *same* HTML content, depending on whether the user is on a mobile phone, tablet, or desktop computer. This latter part is called Responsive Design.



As a web developer, you'll (hopefully) be given a polished design to work off of. Your job is to turn that mockup into a real web page leveraging your knowledge of CSS. As we mentioned earlier, setting individual CSS properties is actually quite simple. The hard part is combining the overwhelming number of built-in properties to create exactly what your web designer asked for—and do it quickly.

This chapter focused mostly on text formatting, but the Cascading Style Sheet language can do a whole lot more. In the next chapter, we'll start exploring how CSS defines the layout of our web pages. On a final note, remember that you can always refer to MDN's CSS Reference when you're not sure how a particular property works.

NEXT CHAPTER >

tutorials. All content is authored and maintained by Oliver James. He loves hearing from readers, so come say hello!

YOU@EXAMPLE.COM SUBSCRIBE

More tutorials are coming. (Lots more.)
Enter your email above, and we'll let you know when they get here.