What might CSS do to improve my website?

John R Hudson

21 September 2019*

During the 1990s there were several attempts to develop style sheets which separated the presentation of material on a website from its content and structure as defined by HTML. Cascading style sheets (CSS) proposed in 1994 by Håkon Wium Lie, who worked at CERN, differed from other attempts in that it allowed styles to be inherited, thereby avoiding the need to create separate style sheets for every page.

They were first adopted by Microsoft in Internet Explorer 3. Later Netscape and Opera also adopted them. CSS2 was published in 1998 but adoption was slow and there were a number of errors in the specification which were remedied through a series of updates between 2007 and 2011 known as CSS2.1. They also went out of favour partly because Microsoft had adopted its customary policy of embracing and extending CSS in ways which were incompatible with what others were doing.

In the middle of the first decade of the 21st century, Mozilla, Apple and Opera began working on a new specification for HTML which would meet the needs of modern devices and which would rely of CSS for styling. This was published in 2011 and work began on a series of rolling updates to CSS to cope with the needs of the new version of HTML and the demands of modern devices.

However, most website generation software today still relies on pre-2011 HTML and so nearly all websites use obsolete HTML and Javascript to accomplish what can be achieved more simply by using post-2011 HTML and CSS.

1 Why use post-2011 HTML and CSS?

1. It is far more readable and easy to maintain than pre-2011 HTML or XHTML.
2. It requires far less code to achieve most effects thereby reducing the burden on browsers and the delay for users accessing a website.
3. The same code can be used to generate a website on a smartphone and a large monitor.
4. Google will prioritise websites which do not have a separate mobile version but display equally well on all devices.

2 A CSS file in use

A CSS file is simply a list of rules to be applied to specific elements in the website.

*First presented at the Manchester Software Freedom Day.

```
body {background-color: mintcream; font: normal 100% serif;}
h1 {font: bold 240% sans-serif; color: darkgreen; font-style: italic; text-align: center;}
h2 {font: bold 130% sans-serif; color: darkgreen; text-align: center;}
h3 {font: bold 110% sans-serif; color: darkgreen;}
header {background-color: palegreen; border: medium solid darkgreen;}
main {display: flex;}
main>article {order: 2;  width: 58%; background-color: mintcream; padding: 1em;}
main>nav {order: 1; float: left; width: 18%; background-color: palegreen;
  border-right: medium solid darkgreen; border-left: medium solid darkgreen; padding: 1em;}
main>aside {order: 3; float: right; width: 18%; background-color: palegreen;
  border-right: medium solid darkgreen; border-left: medium solid darkgreen; padding: 1em;}
footer {background-color: palegreen; border: medium solid darkgreen;}
*.footer {font: normal 75% sans-serif; text-align: center;}
embed {width: 33%; height: 30pt; margin-left: 33%;}
figure.left {float: left;}
figure.right {float: right;}
figcaption {text-align: center; font-style: italic;}
p.labeling {padding-left: 3em; text-indent: -3em; white-space: pre-wrap;}
li {list-style-type: none;}
li.itemi {list-style-type: disc;}
li.enumi {list-style-type: decimal;}
li.enumiind {list-style-type: decimal inside;}
li.enumii {list-style-type: lower-alpha;}
li.enumiirom {list-style-type: lower-roman;}
li.nav {list-style: none; line-height: 150%}
sub, sup {font-size: 60%;}
```

Figure 1: A CSS file

2.1 A CSS rule

A CSS rule consists of a SELECTOR containing one or more comma separated HTML elements[1] to which the rule applies followed by any number of declarations; each DECLARATION consists of a property followed by a colon and one or more values terminated by a semi-colon with all the declarations enclosed in braces

SELECTOR	{DECLARATION;	DECLARATION;	...}
`element, element, ...`	`{property: value value ...;`	`property: value value ...;`	`...;}`

2.2 Some examples

In the example in figure 1, the first line:

```
body {background-color: mintcream; font: normal 100% serif;}
```

contains:

- the element `body`
- the property `background-color`

[1] Space separated elements take on a different meaning.

- the value `mintcream`
- the property `font`
- the values `normal`, `100%` and `serif`.

The HTML element `<body>` contains the content of the page; so this rule sets the background colour for the content of the page and the characteristics of the standard font. Both can be overridden by rules for other elements.

There are around 140 named colours recommended for use because they can be most accurately rendered on most screens; you can specify other colours by using a variety of colour encodings including RGB but you should only do this if there is no suitable CSS colour.

The values for the body text `font` specify an upright `serif` font (like this one) which is the default size for the browser. CSS sizes can be absolute or relative. To ensure that your text is equally readable on a smartphone and a large monitor, only relative font sizes should be specified. Sans-serif fonts have become very popular in recent years but they are more difficult to read than serif fonts when there is a significant amount of text. So choose serif fonts for text rich pages.

You can also specify a value for a `font-family`, such as Times Roman or Helvetica; if you do and the user does not have the font family installed, the browser will substitute something which it thinks is a good match. If you do not specify one, the browser will also select what it thinks is the best option. So specifying a font family may make a difference to all those users who have that particular font family installed and little or no difference to anyone else.[2]

You can specify up to five different heading styles within a CSS heading group; to specify more than five, simply add another group.

```
h1 {font: bold 240% sans-serif; color: darkgreen; font-style: italic;
text-align: center;}
h2 {font: bold 130% sans-serif; color: darkgreen; text-align: center;}
h3 {font: bold 110% sans-serif; color: darkgreen;}
```

Headings are all, by convention, `bold` and `sans-serif`. So the key difference between these three headings is their size, in each case defined relative to the default font size, again to ensure that they have the same relative size whether on a smartphone or a large monitor.

The other, much less important, differences are that the first two are centred and the text in the first one is bold and italic.

The `<header>` element is the block at the top of the page which normally contains the title of the page while the `<footer>` element is the block at the bottom of the page which may contain a variety of links.

```
header {background-color: palegreen; border: medium solid darkgreen;}
footer {background-color: palegreen; border: medium solid darkgreen;}
```

The `border` property has the values `medium` which defines its width, `solid` which determines that it is continuous and a colour (see figure 3).

[2]Most smartphones do not have any of the standard font families installed.

2.3 CSS containers

One of the significant weaknesses of post-2011 HTML when it was first developed was any means of accurately placing elements on a page; for example, though it was recommended to put the `<nav>` and `<aside>` elements on either side of the `<article>` element (figure 2), there was no easy way to ensure reasonably secure placement of these elements on the page. This has been remedied by the provision of CSS containers. Containers can be `flex` or `grid` containers. A `flex` container expands or contracts the elements within it horizontally or vertically so that they fit the available width or height. A `grid` container organises the elements in rows horizontally and vertically to provide the most suitable disposition of elements for the device. So, for example, twelve elements in a `grid` container might appears as two rows of six, three of four, four of three or six of two depending on the dimensions and orientation of the device.

The HTML `<main>` element is normally used to declare a container; the CSS rule specifies whether its `display` property will be `flex` or `grid`.

```
main {display: flex;}
```

The subsequent rules specify that, when a particular element is found within the `<main>` element (in this case), its properties will take the values specified.

```
main>article {order: 2;  width: 58%; background-color: mintcream;
padding: 1em;}
main>nav {order: 1; float: left; width: 18%;
background-color: palegreen; border-right: medium solid darkgreen;
border-left: medium solid darkgreen; padding: 1em;}
main>aside {order: 3; float: right; width: 18%;
background-color: palegreen; border-right: medium solid darkgreen;
border-left: medium solid darkgreen; padding: 1em;}
```

The `order` property specifies the order in which the element will appear, in this case from left to right. So the `<nav>` element is on the left, the `<article>` element comes next and the `<aside>` element comes third.

`<nav>`, `<article>` and `<aside>`, along with `<header>` and `<footer>`, are five of the semantic elements introduced into HTML in 2011 to distinguish the functions of HTML elements within a page (figure 2). One reason for this arrangement was that device screens had become oblong rather than almost square as in the 1990s. But long lines of text are difficult to read because of the distance the eye has to move from the end of one line to the start of the next line. So putting the `<nav>` and `<aside>` elements on either side of the `<article>` element made it narrower and easier to read. It also meant that people opening a page on a smartphone would not have to scroll down to begin to read the main content of the page.

The `width` property specifies the percentage of the width of the page which the element is to take up; again, using relative measurements ensures that the elements will have the same relative size whatever the device.

Only the `border-left` and `border-right` properties are given values for the `<nav>` and `<aside>` elements because the `<header>` and `<footer>` elements already have borders where these elements abut each other.

The `padding` property provides the value of the padding width between the content and the border (figure 3). This prevents the text in the element from going right up to the border. Again, the relative unit — `em` — is used so that the padding is the same relative width whatever the device.

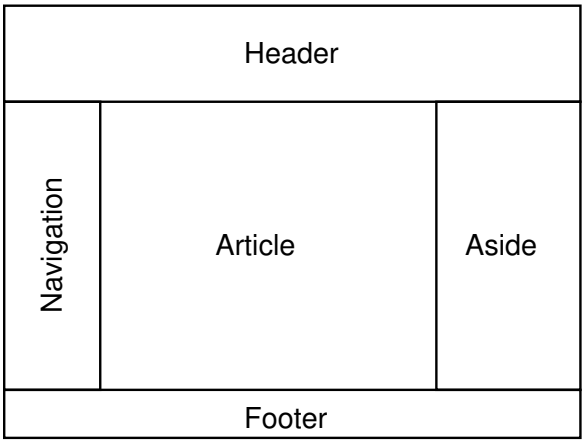

Figure 2: The relationship of five of the elements in the `<body>` element (thanks to David Fisher (2010))

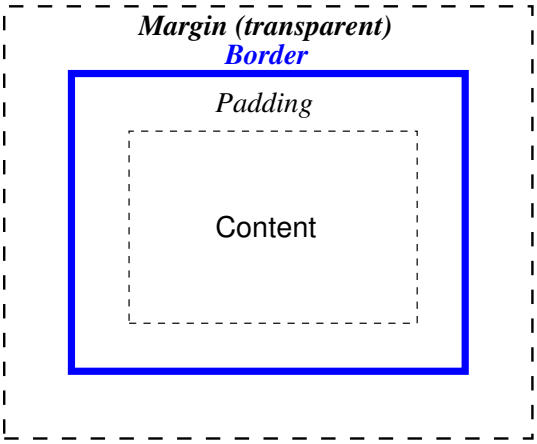

Figure 3: Box model (N.B. the default value of the margin and padding width is 0.)

2.4 Users with visual disabilities

While using CSS containers allows more precise placing of elements on a page, in this example the other purpose is to make the website more screenreader — and therefore visually disabled user — friendly. A browser normally places elements on the page in the order in which they appear in the HTML code; so all the navigation links will normally be read before the content of the page. If a visually disabled user is navigating from page to page they have to hear the same navigation links on every page before they can get to the content of the page.

By specifying the order in which the elements in a CSS container will appear, the `<article>` element can appear immediately after the `<header>` element and before the `<nav>` element in the HTML code so that a visually disabled user can listen to the content of the page and then select where they want to go next while other users will see the elements as in figure 2.

More recently the CSS `voice-` properties have been completely revised so that, if you have dialogue, quotations or verse in a page, you can specify the type of voice the screenreader will use, male or female, young or old, the pitch, the speed, the volume and so on.

2.5 Classes

So far, the rules discussed have applied to defined HTML elements; however, it is also possible to apply the same rule to many different elements or to apply a different rule to different examples of the same element by using the `class=""` attribute in an HTML element.

In the first example, we create a rule which states that any element which has the attribute `class="footer"` will have text which is 75% the size of the default, in a `sans-serif` font and centred.

```
*.footer {font: normal 75% sans-serif; text-align: center;}
```

In the next two examples, we make rules that, when a `<figure>` has the attribute `class="left"`, it will float to the left of the page and, when it has the attribute `class="right"`, it will float to the right of the page.

```
figure.left {float: left;}
figure.right {float: right;}
```

This is particularly useful when adding smaller images to a page as those in the image should always be looking into the page, not out of it. So, if the people in the image are looking to the user's right, put them in a `<figure>` element that floats to the left and *vice versa*.

The next rule is an adaptation of the LyX paragraph style 'labeling' to create a hanging paragraph. The property `padding-left` adds 3 ems of padding at the left of the paragraph; the value `-3em` for the property `text-indent` creates a negative indentation the same width as the padding at the start of the paragraph and the value `pre-wrap` for the property `white-space` prevents the removal of any white space in the indentation.

```
p.labeling {padding-left: 3em; text-indent: -3em; white-space: pre-wrap;}
```

2.6 List styles

The first list style rule below creates a list without bullets or numbers; the next five mimic a number of the possible UK English bullet and enumerated styles in LaTeX.

```
li {list-style-type: none;}
li.itemi {list-style-type: disc;}
li.enumi {list-style-type: decimal;}
li.enumiind {list-style-type: decimal inside;}
li.enumii {list-style-type: lower-alpha;}
li.enumiirom {list-style-type: lower-roman;}
```

However, the number of possible list style types has been greatly extended by the recently introduced `@counter-style` rule which allows the definition of counter style values in a CSS file which can then be used with `list-style-type` properties; a new counter style is defined by

```
@counter-style <counter-style-name> {...;}
```

2.7 Making room for menu items on mobiles

In order to provide additional space between menu items in a `<nav>` element on a smartphone, it is recommended that the `line-height` property be set to 150% at the minimum.

```
li.nav {list-style: none; line-height: 150%}
```

2.8 Keeping subscripts and superscripts small

Without a specific CSS rule, browsers will display subscripts and superscripts the same size as the surrounding text; this rule ensures that they display at something like the recommended size.

```
sub, sup {font-size: 60%;}
```

3 The v unit

The one CSS unit which you may not see in a CSS rule but which you may see in HTML code is the `v` unit, or 1% of the width of the viewport. This is particularly useful when creating a page containing larger images for use on any device. A series of increasingly smaller images are created, the smallest being around 400px, the links to them all being included in a `srcset=" "` attribute with the addition of their width in pixels.

In this example, the `sizes` attribute is set to `sizes="45vw"` so that the image takes up most of the width of the `<article>` element in this particular page, or roughly half the width of the screen, allowing for a margin on either side of the image.

```
<figure>

    <img sizes="45vw" srcset="memories/Rugby_77-78_399px.png 399w,
    memories/Rugby_77-78_600px.png 600w, memories/Rugby_77-78_770px.png 770w"
    src="memories/Rugby_77-78_399px.png" alt="Rugby Team 1977/78" role="img">
    <figcaption>Edwards, Potter, ... </figcaption>

</figure>
```

Browsers that support the `sizes=" "` attribute select one of the sizes in the `srcset=" "` list while the `src=" "` attribute is provided for browsers that do not support the `sizes=" "` attribute.

Use the `Image>Scale image` function in GIMP to create the set of images at 72 pixels to the inch. For very large images, I use 1600, 1200, 800 and 400 pixel images, though there are probably very few devices on which a browser would select the 1600 pixel image — I'm just future-proofing!

References

Fisher, D. (2010, 25 August). HTML5. Talk at the Bradford GNU/LUG meeting at Bradford Council for Voluntary Services, Bradford, West Yorkshire.