CSS for presentation

Cascading Style Sheet Orientation

CSS Cascading Style Sheet is a language that allows the user to change the appearance or presentation of elements on the page: the size, style, and color of text; background colors; border styles and colors; even the position of elements on the page. Presentation, again, refers to the way the document is displayed or delivered to the user, whether on a computer screen, a cell phone display, or printed on paper. With style sheets handling the presentation, HTML can handle the business of defining document structure and meaning, as intended.

Advantages of CSS:

- **CSS saves time** - You can write CSS once and then reuse same sheet in multiple HTML pages. You can define a style for each HTML element and apply it to as many Web pages as you want.
- **Pages load faster** - If you are using CSS, you do not need to write HTML tag attributes every time. Just write one CSS rule of a tag and apply to all the occurrences of that tag. So less code means faster download times.
- **Easy maintenance -** To make a global change, simply change the style, and all elements in all the web pages will be updated automatically.
- **Superior styles to HTML** - CSS has a much wider array of attributes than HTML so you can give far better look to your HTML page in comparison of HTML attributes.
- **Global web standards** - Now HTML attributes are being deprecated and it is being recommended to use CSS. So its a good idea to start using CSS in all the HTML pages to make them compatible to future browsers.

How style sheets Work

It’s as easy as 1-2-3! 1.

1. Start with a document that has been marked up in HTML.

2. Write style rules for how you’d like certain elements to look.

3. Attach the style rules to the document. When the browser displays the document, it follows your rules for rendering elements.

Writing the rules

A style sheet is made up of one or more style instructions (called rules or rule sets) that describe how an element or group of elements should be displayed. The first step in learning CSS is to get familiar with the parts of a rule. Each rule selects an element and declares how it should look.

The following example contains two rules. The first makes all the h1 elements in the document green; the second specifies that the paragraphs should be in a small, sans-serif font.

```
h1 { color: green; }

p  { font-size: small; font-family: sans-serif; }
```

In CSS terminology, the two main sections of a rule are the selector that identifies the element or elements to be affected, and the declaration that provides the rendering instructions. The declaration, in turn, is made up of a property (such as color) and its value (green), separated by a colon and a space. One or more declarations are placed inside curly brackets, as shown in following example:

```
                    declaration                 Declaration block
                         |                              |
Selector   { property: value;  }      selector {
                                                  Property1: value1;
                                                 Property2: value2;
                                                 Property3: value3;
                                                  }
```

Selectors

In the previous small style sheet example, the h1 and p elements are used as selectors. This is called an element type selector, and it is the most basic type of selector. The properties defined for each rule will apply to every h1 and p element in the document, respectively. More sophisticated selectors can be used to target elements, including ways to select groups of elements and elements that appear in a particular context. Choosing the best type of selector and using it strategically—is an important step in using a CSS in appropriate way.

Declarations

The declaration is made up of a property/value pair. There can be more than one declaration in a single rule; for example, the rule for the p element shown earlier in the code example has both the font-size and font-family properties. Each declaration must end with a semicolon to keep it separate from the following declaration. If you omit the semicolon, the declaration and the one following it will be ignored. The curly brackets and the declarations they contain are often referred to as the declaration block. Because CSS ignores whitespace and line returns within the declaration block, authors typically write each declaration in the block on its own line, as shown in the following example. This makes it easier to find the properties applied to the selector and to tell when the style rule ends.

```
p  {
  font-size: small;
 font-family: sans-serif;
}
```

Note that nothing has really changed here—there is still one set of curly brackets, semicolons after each declaration, etc. The only difference is the insertion of line returns and some character spaces for alignment. The heart of style sheets lies in the collection of standard properties that can be applied to selected elements.

Adding comments within a style sheet

You can add comments within a style sheet by using the /* characters to start the comment and the */ characters to end the comment. Comments may also span multiple lines, as shown in the following example.

```
 /* This is the style
 for the body element */
body {
   background-color: white; /* The rgb value is #ffffff */
color: gray; /* This is the font color */
 }
```

Providing measurement values

When providing measurement values, the unit must immediately follow the number like this:

{margin: 2em; }

Adding a space before the unit will cause the property not to work.

{ margin: 2 em;} Incorrect

Also, it is acceptable to omit the unit of measurement for zero values:

{margin: 0; }

Note that 1em is equal to the current font size. 2em means 2 times the size of the current font. E.g., if an element is displayed with a font of 12pt, then 2em is 24 pt. the em is a very useful in CSS, it can adept automatically to the font that the reader uses.

Attaching the styles to the document

There are three ways to style information that can be applied to an HTML document:

Inline styles.

The user can apply properties and values to a single element using the style attribute in the element itself by using the generic syntax:

```
<element style="...style rules....">
```

For example

```
<h1 style="color: red">Introduction</h1>
```

To add multiple properties, just separate them with semicolons, like this:

```
<h1 style="color: red; margin-top: 2em">Introduction</h1>
```

Because an inline style is applied only to the particular element to which you wish to add styling, you don’t need a selector; you just need to specify the declaration block. Inline styles should be avoided, unless it is absolutely necessary to override styles from an embedded or external style sheet. An advantage of using an inline style is that it always overrides styles that are defined elsewhere because the inline styles are specific to the element on which the inline style is defined. This specificity can solve isolated problems when a style is applied globally in an external style sheet, but one element needs to be styled differently.

```
<p style="color: red;">The quick brown fox jumps over  the lazy dog.</p>
```

In the example above, we use a style attribute inside the opening tag. Applying a style to a specific HTML element in this way is known as using an inline style.

```
<p style="color: red; font-weight: bold;">The quick brown fox  jumps over the lazy dog.</p>
```

Notice that a semicolon separates the two declarations. You could carry on adding styles in this way, but beware, this approach can be messy.

Embedded style sheets.

Inline styles offer a simple and quick method to apply some CSS effects to specific sections of a document, but there are better ways to style a page. After all, it would be more ideal if you could set styles in just one place, rather than having to type them out every time you wanted to use them. An embedded style sheet is a section that the user add to the start of a web page that sets out all the styles that will be used on that page. To do this, the style element must be placed in the head of the document and it must contain a type attribute that identifies the content of the style element as “text/css”.

```
<head>
  <title> Simple example of embedded style </title>
          <style type="text/css">
   p {
      font-weight: bold;
    }
      </style>
</head>
```

In the markup shown above, the embedded style sheet starts with a tag. The actual style declarations are enclosed in a set of curly braces: { and }. The p that appears before the first curly brace tells the browser what elements the style rules are for; in this case, the text inside every p will marking as bold. The p is called the selector, and it's a great tool for quickly and easily changing the appearance of lots of elements on the web page. The selector instructs the browser to apply all the declarations between the curly braces to certain elements. The selector, curly braces, and declarations combine to form what's called a rule. In this case, our style sheet contains one rule: "Style all the paragraphs on this page so that the text appears in a bold font." Also it could add more declarations to the rule. For instance, to make the text bold and green, the declaration color: green to the rule:

```
<style type="text/css">
p {
font-weight: bold;
color: green;
}
```

In the example provided, text in all paragraphs will display in bold, green type. This saves the user from typing every time when start a new paragraph—a clear benefit over inline styles. If the user wanted to change the color of all paragraph text to red, only change it in the style sheet at the top of the page is needed. For this reason, an embedded style sheet is a marked improvement over inline styles.

External style sheets.

If the website comprising many pages and the user want to make the changes across the whole site, embedded style sheets falls for a perfect solution because this required to edit the embedded style sheet on every single page of that site. An external style sheet provides a location where you can place styles to be applied on all your web pages.

An external style sheet is a separate, text-only document that contains a number of style rules. It must be named with the CSS suffix. The CSS document is then linked to importe into one or more HTML documents. In this way, all the files in a website may share the same style sheet. This is the most powerful and preferred method for attaching style sheets to content.

To make use of all the benefits of an external style sheet, first need to create a CSS file that can be shared among the pages of the website. Open your text editor and enter the following in a new document:

```
/*
External CSS example
*/
p {
   font-weight: bold;
   color: green;
}
```

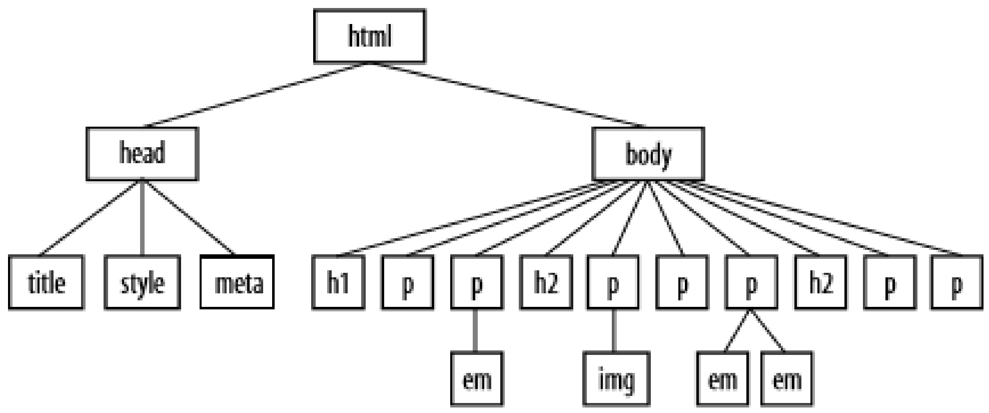
Save the file in the same folder as your HTML files, naming it style1.css; you can save a CSS file in the same way you saved your HTML files. Note that the first few lines we typed into our CSS file won't actually do anything. Like HTML comments where they allow the user to make notes about your work without affecting the onscreen display. For the external CSS example. a rule is added so that all the type in the paragraphs is now bold and green.

The Concepts

There are a few big ideas that need to get a head around to be comfortable with how Cascading Style Sheets behave.

Document structure

This is where an understanding of your document's structure becomes important. In HTML documents have an implicit structure or hierarchy. For example, the sample article that have been playing with has an html root element that contains a head and a body, and the body contains heading and paragraph elements. A few of the paragraphs, in turn, contain inline elements such as images (img) and emphasized text (em). You can visualize the structure as an upside-down tree, branching out from the root, as shown in the following figure.



Inheritance

The HTML elements pass down certain style properties to the elements they contain. For example, when the style of the p elements in a small, sans-serif font, the em element in the second paragraph became small and sans-serif as well, even though we didn't write a rule for it specifically. That is because it inherited the styles from the paragraph it is in.

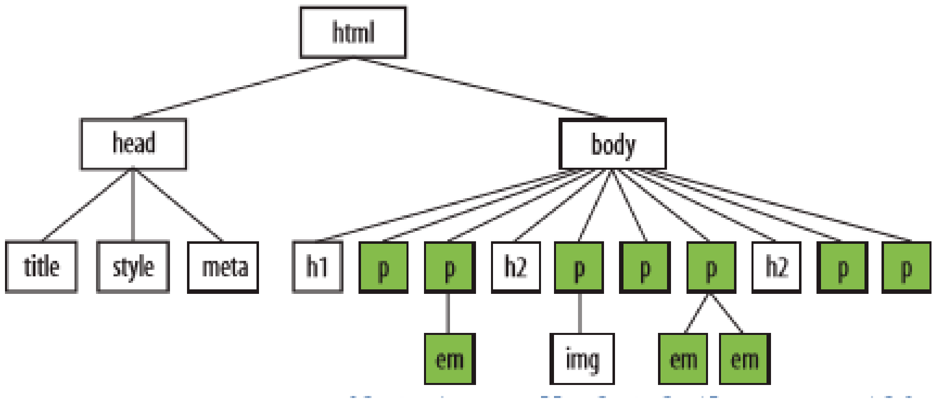
Parents and children

The document tree becomes a family tree when it comes to referring to the relationship between elements.

1- All the elements contained within a given element are said to be its descendants. For example, the h1, h2, p, em, and img elements in the document in previous Figure are all descendants of the body element.
2- An element that is directly contained within another element (with no intervening hierarchical levels) is said to be the child of that element.
3- Conversely, the containing element is the parent. For example, the em element is the child of the p element, and the p element is its parent.
4- All of the elements higher than a particular element in the hierarchy are its ancestors.
5- Two elements with the same parent are siblings.

Pass it on

When you write a font-related style rule using the p element as a selector, the rule applies to all of the paragraphs in the document as well as the inline text elements they contain. We've seen the evidence of the em element inheriting the style properties applied to its parent (p). Also, the figure demonstrates what's happening in terms of the document structure diagram. Note that the img element is excluded because font-related properties do not apply to images



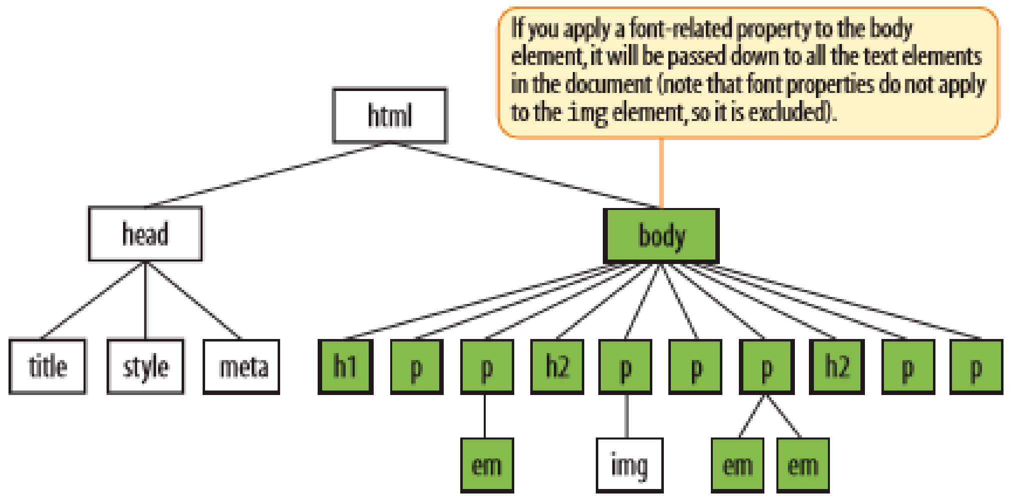
P { font-size: small; font-family: sans-serif; }

Certain properties applied to the p element are inherited by their children

It's important to note that some style sheet properties inherit and others do not.

- In general, properties related to the styling of text—font size, color, style, etc.—are passed down.
- Properties such as borders, margins, backgrounds, and so on, that affect the boxed area around the element tend not to be passed down.

You can use inheritance to your advantage when writing style sheets. For example, if you want all text elements to be rendered in the Verdana font face, you could write separate style rules for every element in the document and set the font-face to Verdana. A better way would be to write a single style rule that applies the font-face property to the body element, and let all the text elements contained in the body inherit that style as in the following figure.


If you apply a font-related property to the body element, it will be passed down to all the text elements in the document (note that font properties do not apply to the img element, so it is excluded).
html
head
body
title
style
meta
h1
p
p
h2
p
p
p
h2
p
p
em
img
em
em

P { font-size: small; font-family: sans-serif; }

All the elements in the document inherit certain properties applied to the body

Conflicting styles: the cascade

CSS allows the user to apply several style sheets to the same document, which means there are bound to be conflicts. For example, what should the browser do if a document's imported style sheet says that h1 elements should be red, but its embedded style sheet has a rule that makes h1s purple?

The designer who wrote the style sheet specification anticipated this problem and devised a hierarchical system that assigns different weights to the various sources of style information. The cascade refers to what happens when several sources of style information vie for control of the elements on a page: style information is passed down ("cascades" down) until it is overridden by a style command with more weight. For example:

- If user don't apply any style information to a web page, it will be rendered according to the browser's internal style sheet (calling this the default rendering)
- If the web page designer provides a style sheet for the document that has more weight and overrides the browsers styles.

Rule order

If there are conflicts within style rules of identical weight, whichever one comes last in the list "wins." Take these three rules, for example:

```
 <style>
p { color: red; }
p { color: blue; }
 p { color: green; }
</style>
```

In this scenario, paragraph text will be green because the last rule in the style sheet—that is, the one closest to the content in the document—overrides the earlier ones. The same thing happens when conflicting styles occur within a single declaration stack:

```
<style>
p { color: red;        color: blue;       color: green; }
</style>
```

The resulting color will be green because the last declaration overrides the previous two. It is easy to accidentally override previous declarations within a rule when you get into compound properties, so this is an important behavior to keep in mind.

The box model

The easiest way to think of the box model is that browsers see every element on the page (both block and inline) as being contained in a little rectangular box. The user can apply properties such as borders, margins, padding, and backgrounds to these boxes, and even reposition them on the page.

To see the elements roughly the way the browser sees them, I've written style rules that add borders around every content element in our sample article.

```
h1 { border: 1px solid blue; }
h2 { border: 1px solid blue; }
p { border: 1px solid blue; }
em { border: 1px solid blue; }
img { border: 1px solid blue; }
```