Web Module 5:

CSS, Part II

Introduction

In this module, the follow-up to Module 5: CSS, Part I, we'll show you how to use CSS to lay out your web pages and then present a smörgåsbord of miscellaneous CSS concepts and techniques that should prove useful for your web styling work.

When you're done working through the module, you'll know:

- How to use CSS to lay out your web pages
- How the box model works
- How to use the CSS float, clear, and position properties to style your s
- How to use CSS to style your web page links
- How to specify values for your CSS properties
- How to create media-specific stylesheets
- How CSS inheritance works
- What's up with the latest CSS standard, CSS3

Caveat Lector

These eight Web modules build on each other. For example, in Module 4 it's assumed you understand the concepts and techniques from Modules 1-3. So, if you run into anything you're unfamiliar with in this module, browse through the earlier modules or turn to Google for help.

Web Page Layout Basics

Every web page you've ever visited in your Internet travels has consisted of a set of rectangular boxes that hold content: text, images, videos, etc. In fact, every block-level element in a web page resides in its own box.

> A block-level element is an HTML element that is displayed in its own line/area in a browser. For example: a <table>, <p> paragraph, or <ul> unordered list.
>
> In contrast, an inline HTML element is displayed in the same line/area as the content that surrounds it. For example: a <b> bolded word or <a> hyperlink in a paragraph.

These boxes are often invisible, with a background color the same as the page's and hidden borders. But, if you look carefully, you can usually make them out by the shape of their content.

Try it with this front page from the NY Times: How many boxes you can find?

I count at least 12. But you can't know for sure unless you look at the page's source code, because boxes can be hidden away, one inside the other, like a set of nested Russian dolls.

Here's a view of the same page using the cool Firefox add-on Tilt that shows the boxes in 3D. Seems I woefully undercounted the number of boxes in this page, which looks closer to 100!

Laying Out Web Pages

Page layout is a key part of the web development process. A poorly laid-out page can severely undermine the clarity and overall usability of the page. And web users love clear, usable pages.

With a big glob of different-sized boxes splattered across a page – see the Tilt screenshot – things can get mightily chaotic in terms of page layout.

Enter the element: THE fundamental building block for laying out web pages.

The (division) element creates containers: boxes that hold other, functionally similar boxes. You use these containers to lay out your web pages.

Historical Note: In the days of webular yore (5+ years back), developers used HTML <table> elements to lay out their web pages, by arranging page content in table cells. This worked okay, but it was basically a kludge, since tables are designed to display spreadsheet-like tabular data, not to be used for page layout. There are still sites out there that use tables for layout, but almost all self-respecting web developers use s these days instead.

Using CSS to Customize Boxes

CSS enables you to exercise very exact control over the appearance and behavior of your boxes – their borders, colors, fonts, margins, sizes, positions, how their content flows, etc. – to achieve the page layout look and feel you are going for.

Note: You can use CSS to control the appearance of all HTML block-level element boxes. But when it comes to page layout, s rule, so we'll focus on them in this module.

Here's a box broken into its four main parts: content, padding, border, and margin.

- Content – the text, images, and other media that the box contains. The content of this is the "Proin gravida ..." text.
- Border – the line around the box. The border of this is displayed as a thick dark-gray solid line. You can use the CSS border property to change these as you see fit. If you don't want a border, set border-width to 0px, or simply omit all border properties from your CSS style for the .
- Padding – the blank space between the content and the border. To change the padding values, you use the CSS properties: padding, padding-bottom, padding-left, padding-right, and padding-top.
- Margin – is the blank space between the border and the surrounding elements on the page: text, images, etc. To change the margin values, you use the CSS properties: margin, margin-bottom, margin-left, margin-right, and margin-top.

In addition to the CSS border, padding, and margin properties mentioned above, here are some commonly used CSS properties for customizing boxes:

- Background properties – background, background-color, background-image
- Dimension properties – height, width
- Position properties – position, top, right, bottom, left (more on these below)

For a full list of CSS properties, see the W3Schools CSS Reference (w3schools.com/cssref).

Explaining the syntax and usage of all the CSS properties mentioned in this module is beyond its scope. Turn to a good CSS Reference like the one at W3Schools for help.

Using HTML s + CSS Styles to Lay Out Your Pages

Advanced page layout using HTML s + CSS styles can get very code intensive and draw on a large range of HTML elements/attributes and CSS style properties. But, with these six HTML and CSS items under your belt, you can create good solid professional-strength page layouts:

- HTML element – defines a division (container, box) in an HTML document
- HTML id and class attributes – names a so you can apply CSS styles to it
- CSS float property – specifies where floating image and text elements appear in a
- CSS clear property – specifies where floating elements cannot appear in a
- CSS position property – specifies where a is positioned on the page

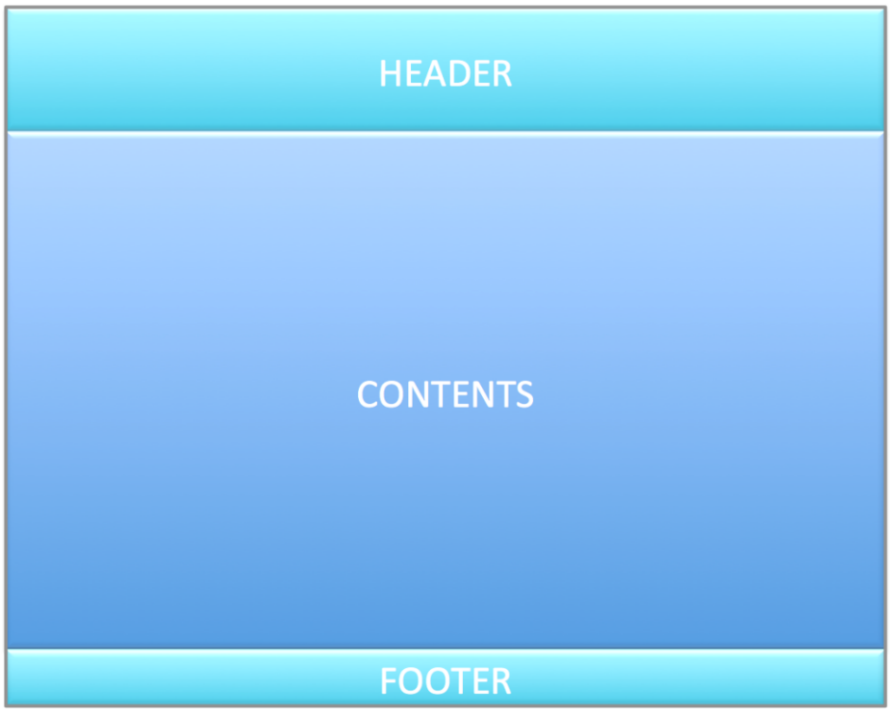
Let's have a look at each of these.

The HTML Element

The HTML element creates an empty box in a page.

s are typically used to divide a page into logical sections for layout. For example, these s create header, contents, and footer sections:

```
<div id="header">...</div>
<div id="contents">...</div>
<div id="footer">...</div>
```



The CSS styles you assign to the header, contents, and footer ids determine the appearance and behavior of their associated s:

```
#header { header-styles }
#contents { contents-styles }
#footer { footer-styles }
```

The HTML id and class Attributes

As you'll recall from the CSS, Part I module:

To apply a style to a single HTML element in a web page – a footer in a page with one footer, for example – you use the id attribute to identify the element in the HTML file:

```
<div id="footer">...</div>
```

and you add a # (hash) before the id name in the CSS style rule:

```
#footer { styles }
```

To apply a style to multiple HTML elements in a page – sidebar s in a page that can have 2+ sidebars, for example – you use the class attribute to identify the HTML element:

```
<div class="sidebar">...</div>
```

and you add a . (dot) before the class name in the CSS style rule:

```
.sidebar { styles }
```

The CSS float and clear Properties

The CSS float property specifies whether an element should float – be pushed to the far left or far right within its box so that other elements can wrap around it. Here's the syntax:

```
selector { float: value; }
```

There are three float values:

- none (default) – the element is not floated
- left – floats the element to the left, wraps other elements around it to the right
- right – floats the element to the right, wrap other elements around it to the left

For example, this style rule floats all images (<img> elements) to the right in a box whose id is set to contents, causing other elements in this to wrap around the images to the left:

```
#contents img { float: right; }
```

The CSS clear property specifies the side(s) of an element where floating elements are not allowed. Here's the syntax:

```
selector { clear: value; }
```

There are four clear values:

- none (default) – allows floating elements on both sides
- left – disallows floating elements on the left side
- right – disallows floating elements on the right side
- both – disallows floating elements on the left or right side

For example, this style rule disallows any floating elements on the left or right side of a <p> (paragraph) element whose class attribute is set to clearfloat:

```
p.clearfloat { clear: both; }
```

To make this style rule apply only to <p> elements in a whose id is set to contents:

```
#contents p.clearfloat { clear: both; }
```

The CSS position Property

The CSS position property specifies a 's positioning on the page, relative to the positioning of the other s on that page. Here's the syntax:

```
selector { position: value; }
```

There are four position values: static, relative, absolute, and fixed. To wit:

static (default)

A static is positioned according to the normal HTML flow of the page, just like it would be if you had not used the CSS position property.

relative

A relative is just like a static , except that you can use four additional properties – top, bottom, left, and right – to reposition it relative to its normal (static) position.

For example, this style rule adds 50 blank pixels above and 100 pixels to the left of where the would appear normally, shifting the 50 pixels down and 100 pixels to the right:

```
#footer { position: relative; top: 50px; left: 100px; }
```

absolute

An absolute is positioned relative to the parent in which it resides. If it doesn't reside in another , the entire web page is taken as its parent.

Other s act as if an absolute were not there. For this reason, an absolute can overlap other elements. You can use the top, bottom, left, and right properties to shift an absolute 's position.

fixed

A fixed is like an absolute , except it's positioned relative to the browser window, not a parent or the entire web page.

Fixed s don't scroll with the rest of the page, which makes them great for things like navigation widgets that you don't want to scroll off the page. You can use the top, right, bottom, and left properties to shift a fixed 's position.

CSS positioning is a challenging topic. Reading about it isn't going to take you very far. It's something you have to practice, over and over, until you get the hang of how it works. To this end, we strongly encourage you to spend some quality time with upcoming Practice task.

The z-index, overflow, and visibility Properties

The z-index, overflow, and visibility CSS properties are used to fine-tune the appearance and behavior of a . Explaining them in detail is beyond the scope of this module. Here's a teaser, just to let you know what these properties are and what they do.

z-index

The z-index property specifies a 's stacking order with respect to other s on the page.

The value of z-index is a number (positive or negative integer).

A with a higher z-index number (integer) is displayed above (in front of) an element with a lower z-index number. It's like layers in Photoshop.

You can use the z-index to create shadows, put text on top of images/text, etc.

This style rule sets the z-index of a with id header to 21:

```
#header { z-index: 1; }
```

z-index only works on non-static s, those whose position property is set to absolute, relative, or fixed. If z-index seems broken, make sure your position is not static (default).

overflow

The overflow property specifies what happens if content overflows a box. Its values are:

- visible (default) – the overflow displays outside the box
- hidden – the overflow is clipped (not visible)
- scroll – the overflow is clipped, but scrollable with a scroll bar
- auto – if the overflow is clipped, a scroll bar is added

Setting overflow to auto is useful for making a in a page whose content scrolls independently from the rest of the page:

```
#sidebar { overflow: auto; }
```

visibility

The visibility property specifies whether a or <table> row/column is visible. Its values are:

- visible (default) – makes the visible
- hidden – makes the invisible
- collapse – removes a row/column from an HTML <table> without changing table layout

visibility is often used together with JavaScript to hide/show s in a dynamic web page:

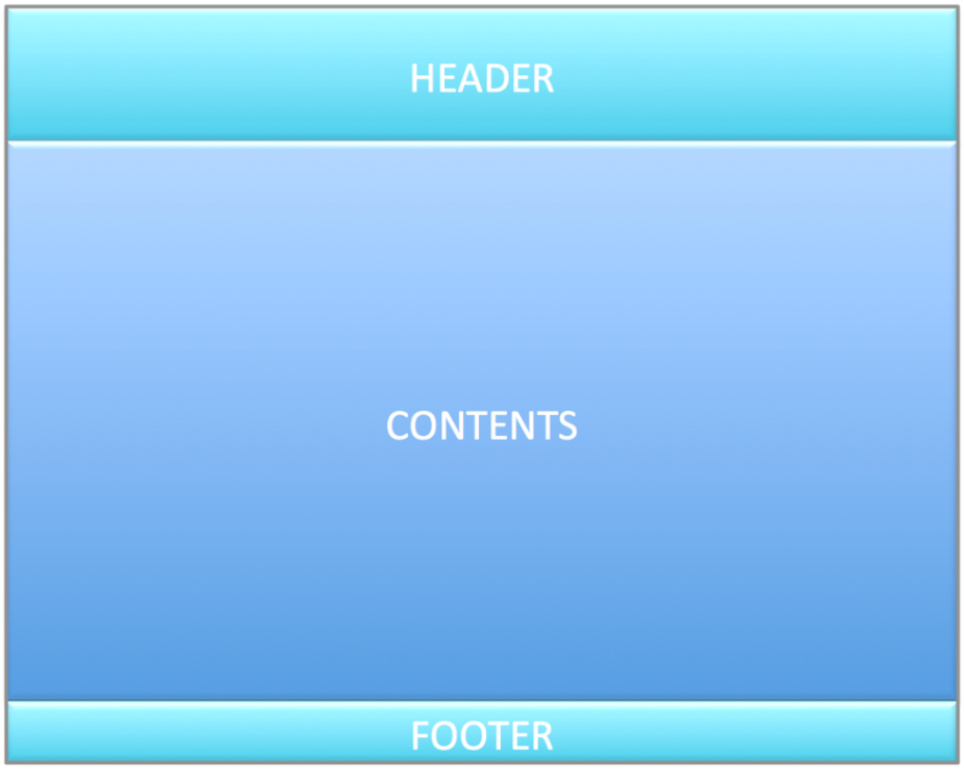
```
#sidebar { visibility: hidden; }
```

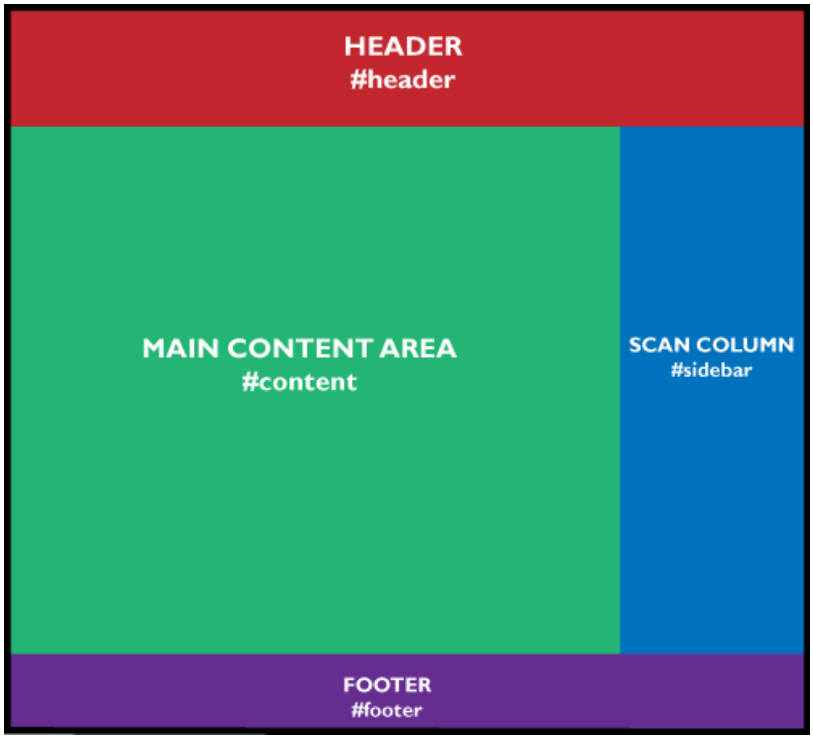
Practice: Using HTML s + CSS Styles for Page Layout

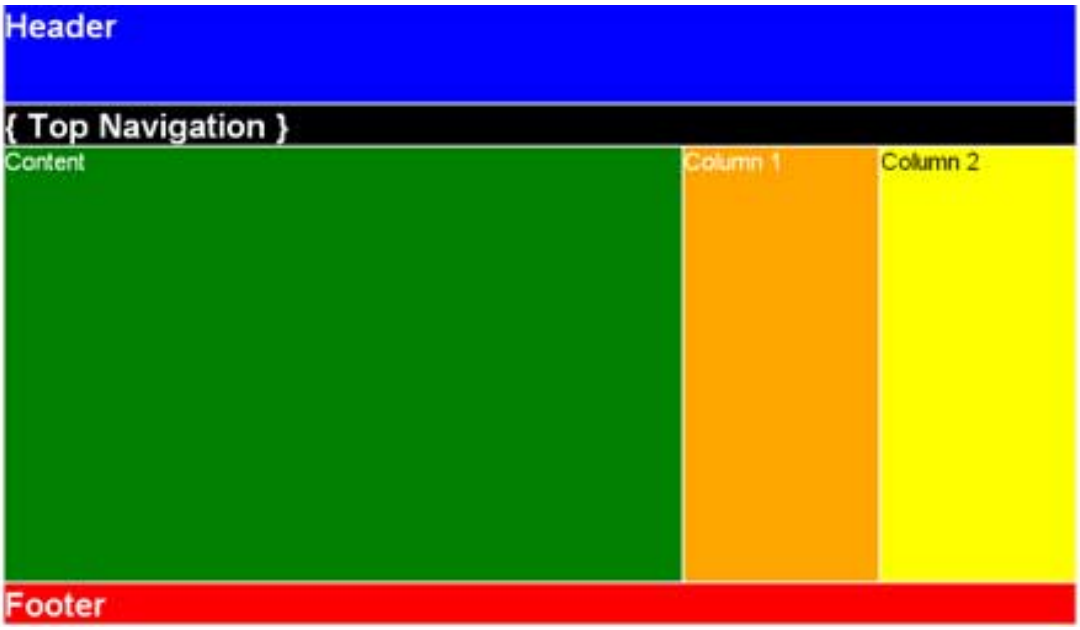
You've been assailed with a ton of CSS concepts and techniques in this module. It's high time to put them into practice!

1. Use HTML s and CSS position properties to create each of the following page layouts.

   Don't worry about matching the colors. Focus on the getting the s positioned correctly with respect to each other and to the page.







2. For each layout, add appropriate content to the s.

   Don't go overboard! Just include enough content to get a feel for what that looks like.

3. Use CSS to style the s in a way that is fits their content: fonts, line spacing, colors, etc.

Using CSS to Style Links

Ever wonder how developers customize the appearance of the links on their web pages? (That default blue underlined look is really boring, isn't it?) How they change a link's color, size, remove the underline, make its appearance change when moused over, etc.

The answer? You guessed it: CSS!

Here's the syntax of the four style rules that you use to customize link appearance:

`a: link { unvisited-link_styles }`

Sets the appearance of an unvisited link.

`a: visited { visited-link_styles }`

Sets the appearance of a visited link.

`a: active {clicked-on-link_styles }`

Sets the appearance of a link being clicked on.

`a: hover { moused-over-link_styles }`

Sets the appearance of a link being hovered (moused) over.

For example, the following CSS code will display a page's unvisited links as red with no underline (text-decoration: none;), visited links as green with no underline, active (clicked-on) links as blue with no underline, and hovered-over links as bold yellow with no underline.

```
a: link { color: red; text-decoration: none; }
a: visited { color: green; text-decoration: none; }
a: active { color: blue; text-decoration: none; }
a: hover { color: yellow; text-decoration: none; font-weight: bold; }
```

Practice: Using CSS to Style Web Page Links

1. Add a handful of links to one of the layouts you created in the previous Practice task.
2. Using the style rules shown above, style the unvisited, visited, active, and hover links.

Challenge: Do something a bit wild with hover, but not so much that it will turn users off.

Specifying CSS Property Values

Whenever you add a property to a CSS style rule, you need to assign it a value:

```
property: value;
```

There are three types of CSS property values:

- Relative values – numerical values that are relative to some other value (20px, 140%)
- Absolute values – numerical values that are absolute (1.5in)
- Named values – values with non-numerical names (red, large, solid)

Relative Values

Normally, you assign relative values to CSS properties:

- px – pixels

  Pixels are relative because displays have different numbers/sizes of pixels.

  ```
  margin: 20px;
  ```

- % – percentage

  ```
  font-size: 140%;
  ```

- em – the size of the current font

  ```
  font-size: 1.4em;
  ```

Absolute Values

There are two main reasons for assigning absolute values to CSS properties:

- For pages to be printed (since the size of a standard printed page is fixed: 8.5x11 inches)
- For pages to be displayed on a device with a fixed screen resolution: # pixels wide x # pixels high (for example: a page shown on a specific handheld device, public kiosk, etc.)

The most commonly used absolute value is:

- pt – point (= 1/72 of an inch)

  ```
  font-size: 20pt;
  ```

These absolute values are less commonly used:

- in – inches
- cm – centimeters
- mm – millimeters
- pc – picas (1 pica = 12 points)

Named Values

You use named values for their convenience and non-numerical simplicity. Some examples:

- medium text is set to the browser's default text size. large text is a certain amount larger.

```
p { font-size: medium; }
h1 { font-size: large; }
```

- A green page background is displayed as the hexadecimal color: #00FF00

```
body { background-color: green; }
```

- solid or dotted or dashed can be used to specify how a border is displayed.

```
#header { border: 1pt solid black; }
#contents { border: 1pt dashed black; }
#footer { border: 1pt dotted black; }
```

For a list of named values for a CSS property, visit the reference page for that property: w3schools.com/cssref

Which Type of Value Should I Assign to My CSS Properties?

Generally, your best bet is to stick with relative values.

> Using px (pixel) values gives your pages the best chance of looking the same (or very similar) on different-sized device screens and in different-sized browser windows.
>
> > If the user is on a high-resolution mobile device (e.g. Retina display), your page might display smaller, but the proportions will be the same.
>
> Many designers use em values for the same reason.

When in doubt:

> Use relative values for pages the user will view: px, %, em.
>
> Use absolute values for pages the user will print: pt.

The Default CSS Stylesheet

Every browser has a built-in default CSS stylesheet that is used to format pages with no custom CSS styles added to them: margins, fonts, alignment, indentation, list bullets and numbers, text decoration (bolded, italicized), etc.

When a page is loaded in a browser, its default stylesheet is applied first to the elements in the page. Afterwards, all the custom styles in the page – stored in <style> elements, HTML tags, or external .css files – are applied.

The moral of the story: Your custom CSS styles override a browser's default CSS styles.

When Styles Conflict, Who Wins?

When two or more styles "compete" for the same HTML element, which style wins?

Let's say you have a page consisting of a single <p> element (paragraph) of text. And let's say that there are three style rules that are competing for this <p> element:

- An external style rule (in a .css file) sets the text color of <p> to red.
- An embedded style rule (in a <style> element) sets <p> text to green.
- An inline style rule (in an opening <p> tag) sets the <p> test to orange?

In what color does the paragraph's text get displayed?

Think it through and take a guess. Then try it out. You might be surprised by the results!

The answer: orange. The inline style rule wins.

When styles conflict, the last style to be processed by the browser wins.

In the above example:

- The very first p style to be processed by the browser is the built-in p style in the browser's default stylesheet.
- The next p style to be processed is either the external p style or the embedded p style, depending on the order in which these appear in the HTML file. Browsers process HTML/CSS code from top to bottom, so the upper line is processed first:

```
<style>p { color: green; }</style>
<link href="styles.css" rel="stylesheet">
```

or

```
<link href="styles.css" rel="stylesheet">
<style>p { color: green; }</style>
```

- The last is the inline p style, because it is processed by the browser right before the paragraph text is displayed:

```
<p style="color: orange;">I win! I win!</p>
```

Media-Specific Stylesheets

Along with creating external .css stylesheets for standard-sized screen displays (desktop, laptop monitor), you can also create stylesheets for other media.

A printer, for example:

```
<link href="printstyle.css" rel="stylesheet" media="print">
```

Using a separate external print stylesheet (printstyle.css), you can lay out and format a page to be printed rather than displayed on a large screen.

Or a handheld device (tablet, phone, watch):

```
<link href="handheldstyle.css" rel="stylesheet" media="handheld">
```

Using an external handheldstyle.css stylesheet, you can lay out and format a page to be displayed on the small screen of a handheld device.

For more information, jump to the Sitepoint page: reference.sitepoint.com/html/link/media

Practice: Creating a Print Stylesheet

1. Use the above instructions to create an external .css stylesheet for a web page that optimizes the page's formatting and layout for printing, rather than viewing.

   You'll probably need to do some online research to get this done; use the Sitepoint link.

2. If you have the chutzpah, create an external handheld stylesheet for handheld devices too.

CSS Inheritance

Inheritance is the mechanism by which style properties of one HTML element are inherited by another element.

For example, if a (bolded) element is embedded in a <p> (paragraph) element, the element (called the descendant) inherits certain style properties of the <p> element (called the ancestor).

Not all properties are inherited!

In general, text-related properties are inherited (color, font, line-height, text-align, etc.), but box-related properties are not (border, margin, padding, etc.).

Inheritance is a key feature in CSS. It prevents developers from having to declare certain properties over and over again, thus streamlining the CSS design process. Pages that use inheritance tend to load faster, which makes for happier clients and lower bandwidth costs.

Inheritance is a deep and complex topic. In this section we'll pretty much just skim the surface. For more, check out this Sitepoint page: reference.sitepoint.com/css/inheritance. Or, if you're feeling ambitious (and brave) turn to the W3C: w3.org/TR/CSS2/cascade.html

Inheriting Style Rules

Say you created this style rule for <p> elements:

```
p { color: red; }
```

How would this paragraph be displayed by a browser?

```
<p>This is <strong>pointless</strong> dude!</p>
```

As all red text, with pointless bolded:

This is **pointless** dude!

This is an example of CSS inheritance in action. The element *inherits* the red color from the p style rule – because the element is nested in a <p> element – but then adds in its own default presentation (bold) to the text that it encloses.

Overriding Inheritance

Given these style rules:

```
p { color: red; }
strong { color: blue; }
```

How would this be displayed in a browser?

```
<p>This is <strong>pointless</strong> dude!</p>
```

Like this:

This is **pointless** dude!

The element *inherits* red from the p style rule, then *overrides* it with blue from the strong style rule.

Not all Properties get Inherited

Given these two style rules:

```
p {
color: red;
border: 1px solid black;
background-color: lightgray;
}
strong { color: blue; }
```

How would this be displayed in a browser?

```
<p>This is <strong>pointless</strong> dude!</p>
```

Like this:

This is **pointless** dude!

The element inherits the background-color rule:

```
background-color: gray;
```

but does not inherit the border rule:

```
border: 1px solid black;
```

Food for thought: How would it look if the element *did* inherit the border rule?

Which CSS Properties get Inherited, Which Not?

1. Reason it out.

   Would it make sense for the nested element inherit the CSS property in question?

   Should pointless have its own border box within the larger paragraph border box?

2. Test it in a web browser.

   Better yet, test it in multiple browsers.

3. Look it up.

   Use the W3C CSS property table: w3.org/TR/CSS2/propidx.html

A CSS3 Teaser

Strictly speaking, the CSS3 specification is still "under development" by the W3C, the organization that comes up with most web standards. However, many of the new CSS3 properties are implemented in newer browsers.

Our advice: Feel free to use CSS3 properties in your web pages. But make sure to test these pages on different browsers – newer and older – to verify that they work correctly, or "fail gracefully" (without turning the page into a sprawling nightmare).

Using CSS3 to Embed Fonts in Web Pages

The CSS3 @font-face selector lets you embed fonts in your web pages.

> An embedded font will display correctly in a user's browser, whether or not that font is installed on their system. This lets you include cool custom fonts in your sites, knowing that your visitors will be able to see them correctly.

For example, this @font-face style rule embeds the custom font Godzilla_lite in a web page:

```
@font-face {
  font-family: Godzilla_lite;
  src: url(godzilla_lite.ttf);
}
```

For more on @font-face usage, turn to W3Schools (w3schools.com/cssref/css3_pr_font-face_rule.asp) or another reputable CSS reference site.

Here are some sites from which you can download free fonts for embedding:

- Font Squirrel – fontsquirrel.com
- Google Fonts – google.com/fonts
- Adobe Typekit – typekit.com/fonts

CSS3 Resources

Here are some juicy CSS 3 resources for your stylistic edification:

- caniuse.com – compatibility tables for HTML5 and CSS3 in desktop/mobile browsers
- css3.info – "Everything you need to know about CSS3"
- w3.org/TR/css3-roadmap – W3C CSS3 roadmap
- w3.org/TR/css3-selectors – W3C CSS3 selectors
- w3schools.com/css/css3_intro.asp – W3Schools CSS3 intro

Here are a few CSS3 (and 2.1) gallery sites to whet your CSS-ing appetite:

- CSS Design Awards – cssdesignawards.com
- CSS Winner – csswinner.com
- CSS Zen Garden – csszengarden.com
- cssline – cssline.com
- cssmania – cssmania.com

And here's a useful page that lists the browser support for all CSS3 properties:

- CSS3 Browser Support – w3schools.com/cssref/css3_browsersupport.asp