Introduction to CSS

Cascading Style Sheets (CSS) are used to style or create layout for HTML documents. In the first part of this course we saw how to use HTML to create document structure, this course will cover the basics of using CSS to apply formatting to structured HTML content.

CSS Style rules

CSS style sheets are defined using rules that have two parts: a selector and a declaration, which is composed of pairs of properties and their associated values. The selector indicates the part of the document to which styles should be applied. The declaration contains the CSS properties and the values to be assigned to them and takes the following form:

```
Selector {property: value;}
```

The declaration, must be enclosed in curly braces. Each property is separated from and its value by a colon; property value pairs must end with a semi-colon.

Selectors

Selectors indicate which part of a document will be formatted. There are three basic types of selectors: HTML, class and ID.

HTML Selectors

The simplest type of selector is the HTML selector which targets a specific HTML element: In the example below, the selector is a paragraph, indicated by the element p; the value *bold* after the property *font-weight* indicates that the text in each paragraph will be bold.

```
p {font-weight: bold}
```

Class selectors

Classes allow styling to be applied independent of an HTML element. They are applied to HTML elements using the class attribute in an element's start tag. Class style rules must be preceded by a period. The examples below show the format for a class style rule called *bodycopy* and a paragraph element to which that class has been applied.

```
.bodycopy {property:value;}
```

```
<p class="bodycopy">paragraph's text goes here</p>
```

Dependent classes. These are class styles specified for one particular HTML selector. They are defined using the format *selector.classname* and will only be applied when that class is assigned to the specified HTML element.

```
p.copy {property:value;}
```

```
<p class ="copy">paragraph's text goes here</p>
```


ID selectors

Similar to class selectors in that they are independent of any specific HTML element, ID Selectors are used to identify a unique item in a document. They are defined in the head section, the rule is preceded by a hash mark:

```
#uniqueItem {property: value;}
<div id="uniqueItem">content goes here</div>
```

When applied to an HTML element, an ID selector turns it into an object. Unlike regular (unnamed) HTML elements, objects can be manipulated with JavaScript, a scripting language that web developers use to create browser (client-side) interactivity: buttons that change when the mouse cursor hovers over them, drop down or collapsing menus, web alert forms. This is one reason why an ID selector can only be applied to one instance of an element per document. If two tags use the same ID selector a browser will not know which tag is being reference by JavaScript, leading to a syntax error and failure of any client-side interactivity.

Although ID selectors can be applied to most HTML elements, they are usually applied to the DIV

Group selectors

If style rules are to be assigned to multiple selectors, they can be defined using a comma-separated series of selectors with a common declaration. The style rule will be applied to any of the selectors.

```
h1, h2, h3 {property: value;}
```

HTML selectors, classes and ids can be part of the same group

```
p, #id_name, .class_name {property: value;}
```

Descendant selectors (sometimes called Contextual selectors)

Descendant selectors target child elements in a specific context. Their style rules will be applied only when the elements in the selector are found in the specified relationship. The selectors in Descendant selectors are separated by spaces, not commas as they are in group selectors. This is a very important distinction. You will get very different results depending on whether a space or a comma separates selectors.

```
ol li {font-style: italic;}
```

In the CSS contextual style rule above italic formatting will only be applied to list items <li> that are inside of an ordered list. List items inside of an unordered list will not be affected. As you can see, Descendant selectors target hierarchical relationships and must be constructed in accordance with HTML's syntactical rules governing parent/child relationships. A parent selector is always placed before the child selector.


```
Good: p b {font-color: red;}
Bad:    b p {font-color: red}
```

In the first example above (Good), the bold tag will create red text only when it is a child of a paragraph element and not when it is a child of any other block element. Because HTML rules do not allow an inline element such as Bold to be a parent of the paragraph element. The second example above (Bad) is wrong

The example below is a more sophisticated and powerful use of this selector.

```
ol ul li {font-family: arial}
```

In the example above the content of a list item will be formatted in Arial only if it is a child of an unordered list which in turn is a child of an ordered list. This type of selector solves the problem of how to target formatting to common child elements in a specific context or relationship with their parents without creating style rules for each individual situation.

Span and Div

Span and Div are used to create custom elements that are independent of other HTML elements. Span creates inline formatting; div creates block formatting.

Div elements are usually assigned an ID selector using the id attribute which turns it into an object that can be referenced with JavaScript. Span elements are usually assigned class selectors to create inline formatting.

Inheritance

HTML elements have a parent/child relationship. An element that is contained in another element is the child of that element. Styles applied to specific elements are inherited by all their child elements. Any element placed inside of a specific element or parent will inherit properties of the parent as long as they don't conflict with its own style rules.

For example, a few words in a paragraph are to be made bold using the <b> element. Style rules for both the bold and paragraph HTML selectors are shown below. The bold text will inherit all the CSS properties assigned to its parent, the paragraph. However because the child element, b, has its own color property it will not inherit color from its parent. The paragraph selector's color, blue, will be replaced with red as defined in the style rule for the bold selector.

```
p {color: blue; font-size: 14px; font-family: arial;}

b {color: red;}

<p>I have a specific font size, font face and color (blue). <b>I also have my own specific color (red) but inherit my font size and font face from the paragraph selector since those properties are undefined for me.</b> See the CSS rules above.</p>
```


Style rule Placement

Style rules can be written in or assigned to a document in three ways: as an inline rule, in a line of text; as an embedded rule, in the head section of a document; or in an external document, called a stylesheet, which is linked to the current document.

Inline rules

These are written in a specific HTML element using the style HTML attribute. Inline rules must be entered in each element to be formatted. This is a useful way to change one specific element, but is not an efficient way to format a number of elements.

Format

```
<element style="property: value; property: value">
```

Example

```
<span style="font-size: 14px; font-weight: bold;">text goes here</span>
```

Embedded style rules

Defined in the head section of a document embedded style rules are available to all elements in the document. They must be written within the style element as shown below.

```
<style type="text/css">
        selector {property: value; property: value }
</style>
```

External style rules

External style rules are written in a plain text document called a stylesheet, which is independent from any HTML documents. A style sheet document is usually saved with the .css extension, using this extension is a good practice which is commonly used but not required. Because many HTML documents can be linked to one or more style sheets, using external style sheets are an efficient way to style a number of documents: define the style rule once and apply to many documents. To change a style you only have to edit it in the external stylesheet, not on each HTML page.

Linking to external style rules. There are two ways to link to an external style sheet: using link element or @import rule. Older browsers prefer the <link> element when importing an external StyleSheet into an HTML document. Below is the format for a link to an external stylesheet.

```
<link rel="stylesheet" href="stylesheet.css" type="text/css">
```

Imported style rules. To import a stylesheet, place the import rule in the style tags in the head of a document. (If placed outside of the style tags, a browser may render it as text in the body of the document.)

In the style element enter:

```
<style type="text/css">
        @import url(path/filename.css);
</style>
```

Like any other declaration @import must end with a semicolon (;), whether it is placed in an HTML file or is part of an external stylesheet. Yes, stylesheets can import other stylesheets.

Setting up an External StyleSheet

To create an external stylesheet, place the style rules in a plain text document which has no HTML tags. If there is HTML coding in the stylesheet it will fail. Common names for a site-wide stylesheet are: global.css or globalstyles.css

Cascade order

There are times when more than one style rule will be applied to an element. If there is a conflict, Cascade order determines how these style conflicts will be resolved. In general, a more specific style rule takes precedence over a more general one. For example, embedded style rules are applied in the order they are written, rules that are lower in the listing will override rules that precede them.

Rules order of precedence:

1. The most specific rule takes precedence: Inline rules override embedded rules which in turn override external rules.
2. ID styles override class styles which in turn override HTML selector styles.
3. !Important trumps all.

When defined in embedded styles and external stylesheets, Grouped Selectors should precede specific rules for an individual HTML selector, class, or id.

```
h1, h2 {color: red; font-family: arial;}
h1 {font-size: 18px;}
h2 {font-size: 16px;}
```

!important

The value !important in a style rule indicates the associated value cannot be overridden by another declaration that would normally take precedence. For example:

```
p {color: red !important; font-size: 12px;}
<p style="color: blue; ">text goes here</p>
```

In the example above the inline style rule which assigns the color blue is prevented from overriding the red color assigned to the ‘p’ selector by the !important argument.

Notice in the example above how the !important argument is place between the value red and the semicolon. Also notice that it is separated from the value by a space. Actually !important is another value. It says that any value(s) that are assigned to this property for this ‘p’ selector cannot be overridden under any circumstance. The value red has top billing or the highest possible precedence.

Media types

CSS can be targeted for different media. It is not uncommon to create one stylesheet for screen presentation and another for document printing. Light type on a dark background is often used for the screen presentation, however it is not efficient for the printed page. Using separate style sheets, one can reverse the onscreen look for printed output.

Screen: for web or presentation
Print: for printable web pages
Below is an example of two linked style sheets, one for screen display and one for print output:

```
<link rel=”stylesheet” href=”path/filename.css” media=”print” />
<link rel=”stylesheet” href=”path/filename.css” media=”screen” />
```

The order in which these stylesheets are coded is critical: if a browser doesn’t understand the media attribute it will use the cascade order to determine which stylesheet to use.

Other media types (currently not widely supported):

- aural (for speech)
- braille
- projection
- handheld

