# SISTA 230

**Cascading Style Sheets (CSS)**
Making Things Pretty

## Review

- What is purpose of HTML?
    - To provide structure, organization, and meaning

## HTML

### The bad old days...

Before CSS, web designers would have to use the HTML 'align' attribute to center-align a paragraph:

**HTML**

```
<p align="center">

      Happy Valentine's Day!

</p>
```

## HTML

### The bad old days...

Similarly, web designers would have to use the <font> tag with a 'color' attribute to specify that the text should be red:

HTML

```
<p align="center">
   <font color="red">
      Happy Valentine's Day!
   </font>
</p>
```

This approach required adding additional information to the HTML that **didn't** provide structure, organization, or meaning. While this was bad from a semantic standpoint, it also had ramifications for maintaining web sites.

The addition of extra HTML tags and attributes cluttered the source code for web pages, making it more difficult to read. Additionally, updating a website's look and feel required making edits on every single page on the site.

## Review

- What is purpose of HTML?
  - To provide structure, organization, and meaning
- The purpose of CSS is to guide the display of the content

## CSS

- CSS - Cascading Style Sheets
- Allow us to specify display properties of an element
- Default styles are provided by the browser
  All web browsers use a CSS file to specify everything from the default font on web pages to the amount of spacing between paragraphs.
- We can override browser-specified styles

## CSS

### CSS Rule Syntax

All CSS rules follow the format of

propertyName: properyValue;

For example, here are CSS rules for specifying the font color, the text alignment, and the background color for an element.

CSS

```
color: red;
```

```
text-align: left;
```

```
background-color: blue;
```

## CSS

### CSS Implemented: At the element level

CSS rules can be specified at the element level by putting them inside of an elements' style attribute.

**HTML**

```
<p style="color:red">
   Happy Valentine's Day
</p>
```

**Browser**

**Happy Valentine's Day**

## CSS

### CSS Implemented: At the element level

You can apply multiple CSS rules to a given element by entering them into the style attribute, separated by a semicolon.

**HTML**

```
<p style="color:red; text-align:center;">
   Happy Valentine's Day
</p>
```

**Browser**

**Happy Valentine's Day**

This method of adding CSS rules doesn't provide us with much improvement over the previous methods of using HTML tags for formating. Not much is gained with regards to simplifying maintenance. For example, we would still have to make multiple changes to a page if we wanted to change all of the red text to blue.

All that said, this approach does provide some simplicity in that it is very clear which elements are effected by the CSS rules.

## CSS

### CSS Implemented: Within the page head

Alternatively, you can specify sets of CSS rules at the page level by utilizing the <style> element (always placed within the <head> tags).

The format for doing this uses CSS selectors (covered later in more detail) and sets of CSS rules. For example, if one wanted to specify the font color and text alignment for all paragraphs (<p> tags), you would enter the following:

**HTML**

```
<head>
   <title>My page Title</title>
   <style type="text/css">
     p
     {
         color:red;
         text-align:center;
     }
   </style>
</head>
```

Using the method described above, we can apply a set of CSS rules to multiple elements on the same page. This allows us to modify multiple elements while only entering the CSS rules once.

While this does provide us with some improvement over using the 'style' attribute, it is still limited by the fact that one would need to enter the same set of CSS rules on every page if they wanted to maintain a consistent look and feel on every page of their website.

## CSS

### CSS Implemented: External CSS File

The preferred method for applying CSS rules to your web pages is to put the CSS selectors and rules in a separate file and then 'link' to that file from each page that you want to apply those rules to.

To do this, one needs to use the <link> element (always placed within the <head> tags), with a *type* attribute of 'text/css', a *rel* attribute of 'stylesheet', and an *href* attribute specifying the URL for the CSS file.

**HTML**

```
<head>
   <title>My page Title</title>
   <link type="text/css" rel="stylesheet" href="cssFile.css" />
</head>
```

**CSS - cssFile.css**

```
p
{
   color:red;
   text-align:center;
}
```

The format for using an external CSS file is similar to that of using a <style> tag on each page. Using CSS selectors (covered later in more detail) and sets of CSS rules, one can specify how elements should appear on any page that links to the given CSS file.

## CSS

### CSS - Comments

CSS

```css
/* You can also add comments to your CSS statements */
body
{
   background-color:purple;
}

p
{
   color:red;
}
```

CSS

```css
body
{
   background-color:red;
}

/*
 * Comments can span multiple lines
 * if you so desire!
 */
p
{
   color:red;
}
```

As with HTML files, comments are always useful for documenting why things were done in a specific way. This will make editing and/or debugging in the future much easier for you and any other developers who may use your code.

## CSS

### The bad old days...

Most CSS properties can be implemented using HTML attributes...

**HTML**

```
<body bgcolor="red">
```

**CSS**

```
body { background-color:red }
```

**Don't use HTML attributes to format the display of your content!**

## CSS

### Selectors

- Used to specify what elements to apply a set of CSS properties to.
  The format for using CSS selectors is to specify the selector, followed by a left curly brace, followed by a set of CSS rules, followed by a closing right curly brace.

As with HTML, multiple spaces are viewed as a single space. In other words, you can use white space to help organize your CSS files, knowing that it will not impact the way the page is displayed.

## CSS

### Element Selector

E - Matches any E element

CSS

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

## CSS

### Element Selector(s)

You can apply a set of CSS rules to multiple elements

CSS

```css
h1, p
{
   color:red;
}
```

You can apply a set of CSS rules to multiple selectors by separating the selectors with a comma.

## CSS

### Universal Selector

* - Matches all elements

CSS

```css
*
{
   color:red;
}
```

The universal selector is good to know about but doesn't often come into use, as it is generally a better approach to be specific with your CSS selectors.

## HTML

### Class and ID attributes

- Can be specified for all HTML elements
- Useful for targeting an element or group of elements for styling

## HTML

### ID attribute

- Each element must have a unique value

**HTML**

```html
<p id="book1">Robinson Crusoe</p>
<p id="book2">Lord of the Flies</p>
<p id="book3">Twilight</p>
```

Id attributes are useful for providing a unique identifier for a given element. Because it can only be used once on a page, id attributes should be specific and provide additional semantic information about the contents or purpose of the element.

## HTML

### ID attribute

- This is invalid!
  Remember, a given value can only be used in one id attribute per page.

**HTML**

```html
<p id="book">Robinson Crusoe</p>
<p id="book">Lord of the Flies</p>
<p id="book">Twilight</p>
```

## HTML

### Class attribute

- Can be specified for all HTML elements
- The same value can be shared with multiple elements
- A single element can have multiple class values (but only one attribute)
  Class attributes are the counterpart to 'id' attributes. Whereas a value can only be used for one *id* attribute on a given page, a *class* attribute can be used multiple times to group similar elements. For example, on a page with multiple paragraphs, all containing information about a book, we might use the following:

**HTML**

```html
<p class="book">Robinson Crusoe</p>
<p class="book">Lord of the Flies</p>
<p class="book">Twilight</p>
```

## HTML

### Class attribute

- Class value(s) must start with a letter and be made up of the following:
    - letters
    - numbers
    - underscores
    - dashes

## CSS

### Class Selector

.myClass - Matches all elements that have a class attribute of 'myClass'

**HTML**

```
<p class="book">Robinson Crusoe</p>
<p class="author">Daniel Defoe</p>
```

**CSS**

```
.book
{
   color:red;
}
```

**Browser**

**Robinson Crusoe**

**Daniel Defoe**

## CSS

### Class Selector

.myClass - Matches all elements that have a class attribute of 'myClass'

An element can have multiple class attribute values. If CSS rules are specified for more than one class and a given element matches both selectors, then both sets of CSS rules may be applied.

**HTML**

```html
<p class="book centered">Robinson Crusoe</p>
<p class="author">Daniel Defoe</p>
```

**CSS**

```css
.book { color:red; }

.centered { text-align:center; }
```

**Browser**

**Robinson Crusoe**

**Daniel Defoe**

## CSS

### Class Selector

E.myClass - Matches all elements **of type E** that have a class attribute of 'myClass'

HTML

```
<p class="book centered">Robinson Crusoe</p>
<p class="author">Daniel Defoe</p>
```

CSS

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

strong.centered { text-align:center; }
```

Browser

```
Robinson Crusoe
Daniel Defoe
```

## CSS

### Class Selector

E.myClass - Matches all elements **of type E** that have a class attribute of 'myClass'

Using combined CSS selectors, you can specify elements that have more than one class value.

**HTML**

```
<p class="book centered">Robinson Crusoe</p>
<p class="book">Lord of the Flies</p>
```

**CSS**

```
p.book.centered { background-color:blue; }
```

**Browser**

**Robinson Crusoe**

**Daniel Defoe**

## CSS

### ID Selector

#myID - Matches the element that has an id attribute of 'myID'

Remember, since a given value can only be used in one id attribute per page, the styles applied using ID selectors will be very specific.

**HTML**

```html
<p id="websiteAuthor">My Name</p>
```

**CSS**

```css
#websiteAuthor { color:red; }
```

## CSS

### ID Selector

**HTML**

```
<p id="websiteAuthor">My Name</p>
   .
   .
   .
<p id="websiteAuthor">My Name Again</p>
```

**CSS**

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

Because the value 'websiteAuthor' is used for more than one element's *id* attribute, the above HTML is not valid. That said, browsers will still apply the CSS rules to both elements. That said, just because something can be done does not mean it should be done. Always validate your HTML!

## CSS

### ID Selector

HTML

```
<p id="websiteAuthor">My Name</p>
```

CSS

```
strong#websiteAuthor { color:red; }
```

The CSS rule would not be applied to the paragraph element in the example above.

## CSS

### ID Selector

**HTML**

```
<p id="websiteAuthor">
   My name is <strong id="websiteAuthor">Justin</strong>
</p>
```

**CSS**

```
strong#websiteAuthor { color:red; }
```

This rule would not be applied to the paragraph element in the example above.

Again, this is not valid HTML because the value 'websiteAuthor' is used for more than one element's *id* attribute.

## CSS

### ID Selector

HTML

```html
<p id="websiteAuthor">
   My name is <strong id="websiteAuthorName">Justin</strong>
</p>
```

CSS

```css
strong#websiteAuthor { color:red; }
```

This CSS rule would not be applied to any elements.

## CSS

### ID Selector

HTML

```
<p id="websiteInfo">
   My name is <strong id="websiteAuthor">Justin</strong>
</p>
```

CSS

```
strong#websiteAuthor { color:red; }
```

This rule would not be applied to the <p> element in the example above but would be applied to the <strong> element.

## CSS

### Combining Class and ID Selector

Just as we can combine element and class selectors, we can combine id and class selectors as well.

#myID.myClass - Matches the element that has an id attribute of 'myID' **AND** has a class attribute of 'myClass'

**HTML**

```
<p id="websiteAuthor" class="redText">My Name</p>
<p class="redText">Phone Number</p>
```

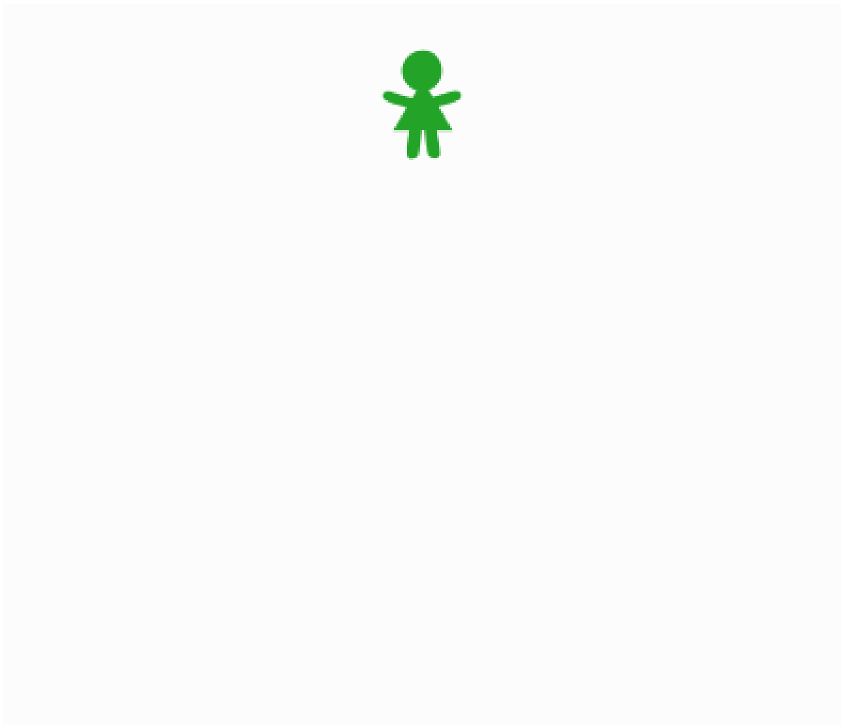
**CSS**

```
#websiteAuthor.redText { color:red; }
```

This rule would be applied to the first <p> element in the example above but not the second.

## HTML

### Document Object Model (DOM)

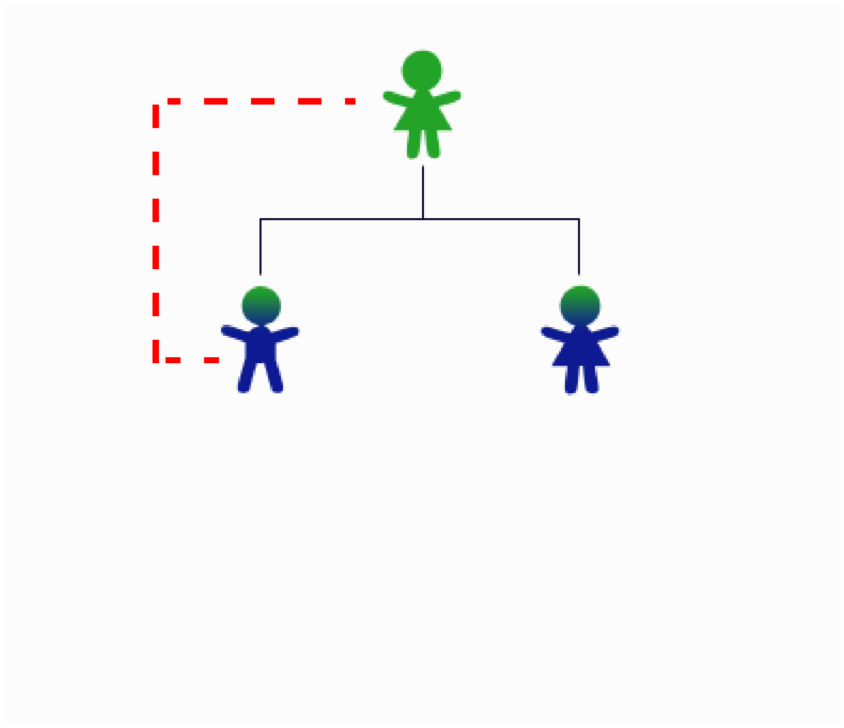


In order to understand the relationship between elements, it is important to ensure that we understand the set of relationship types available to us. Consider the example of a family tree.

## HTML

### Document Object Model (DOM)

Parent/child relationship

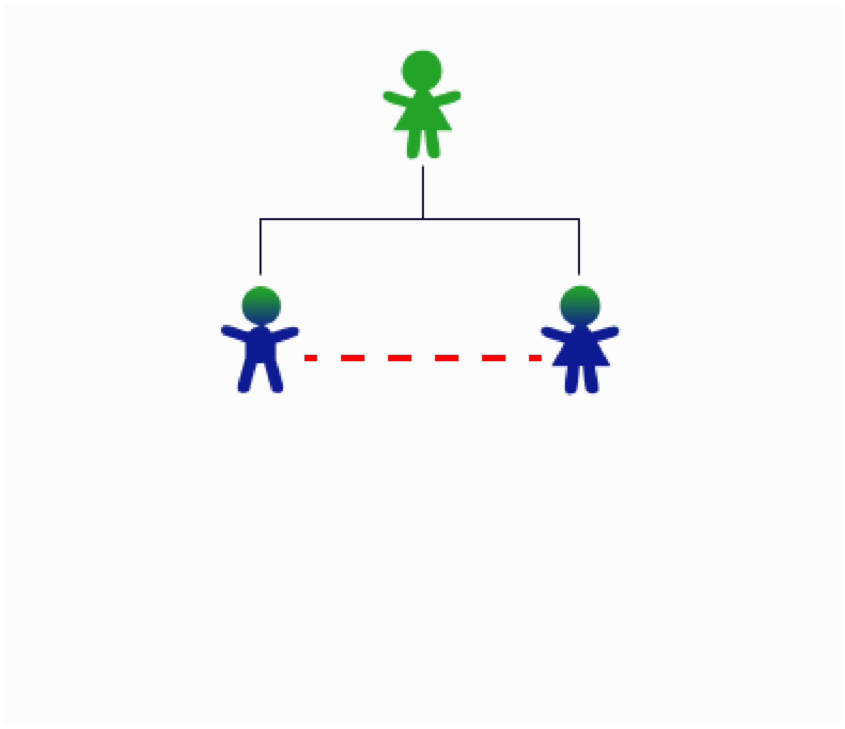


There is a relationship between a parent and a child.

## HTML

### Document Object Model (DOM)

Sibling relationship

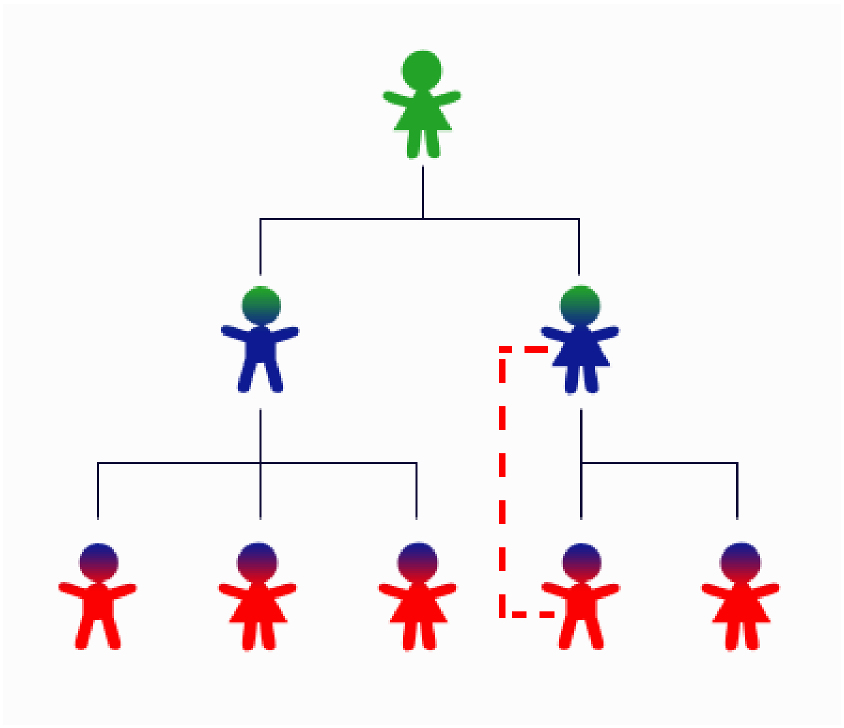


There is a relationship between siblings (i.e, children who share the same parent).

## HTML

### Document Object Model (DOM)

Parent/child relationship

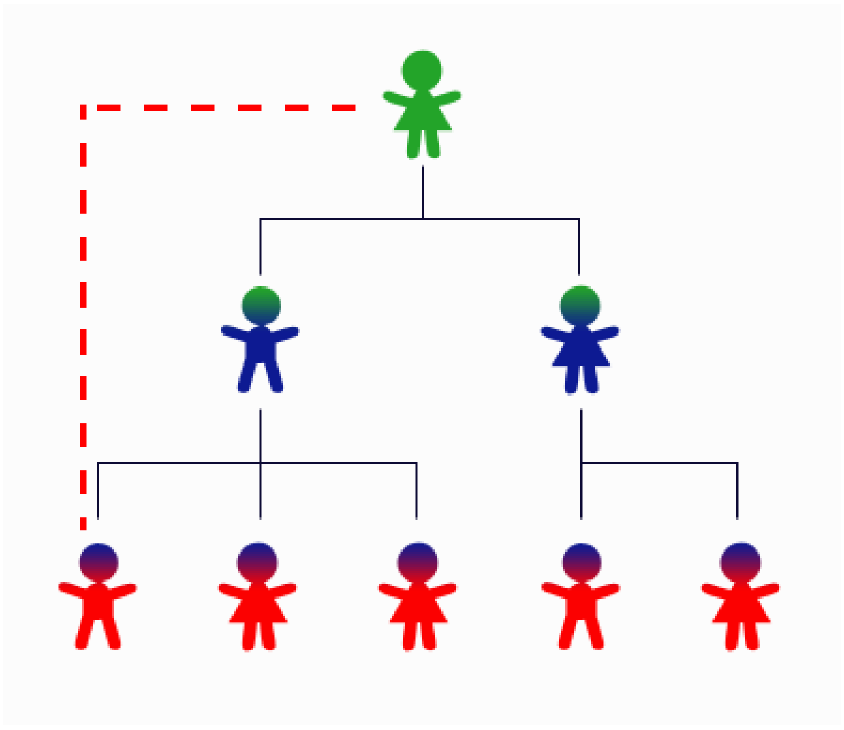


Children can also have their own children, initiating a parent/child relationship of their own.

## HTML

### Document Object Model (DOM)

Ancestor/descendant relationship

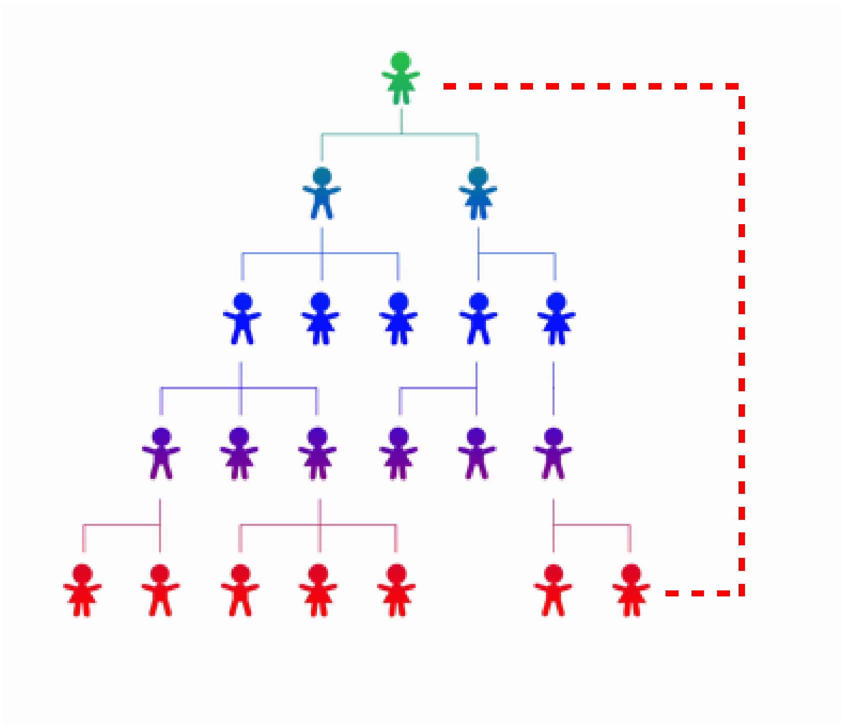


Rather than defining things at a granular level (e.g., grandparent, great-grandparent, etc.), we simply use ancestor/descendant to describe any relationship wherein one person is a parent of another elements parent, and so on.

## HTML

### Document Object Model (DOM)

Ancestor/descendant relationship



## HTML

### Document Object Model (DOM)

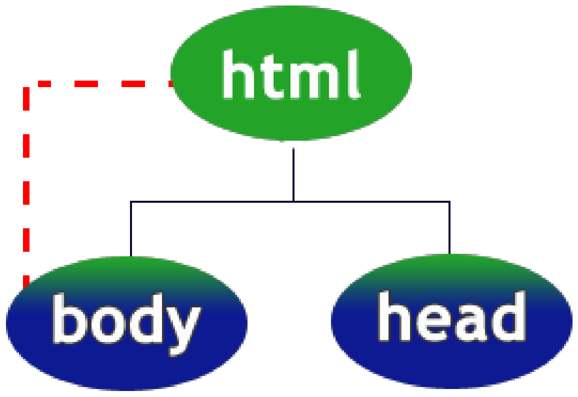
- Document Object Model (DOM)
  - Way of representing an (X)HTML document
  - Defines relationships between elements

The DOM uses the family tree analogy to define the relationship between elements in an HTML page.

## HTML

### Document Object Model (DOM)

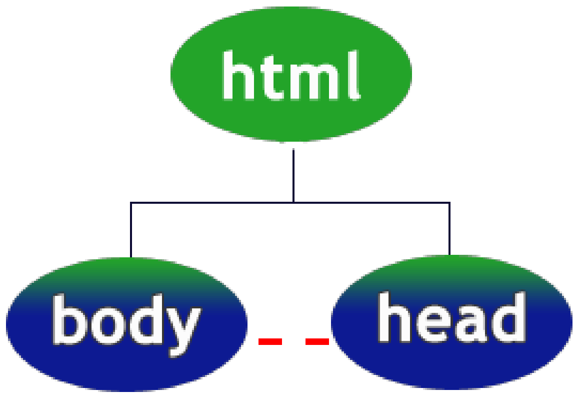
Parent/child relationship



## HTML

### Document Object Model (DOM)

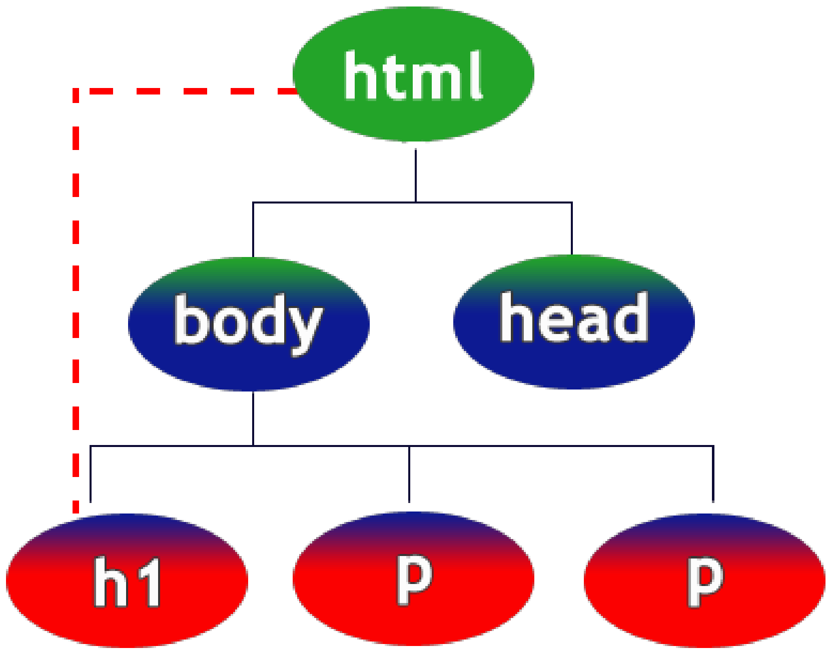
Sibling relationship



## HTML

### Document Object Model (DOM)

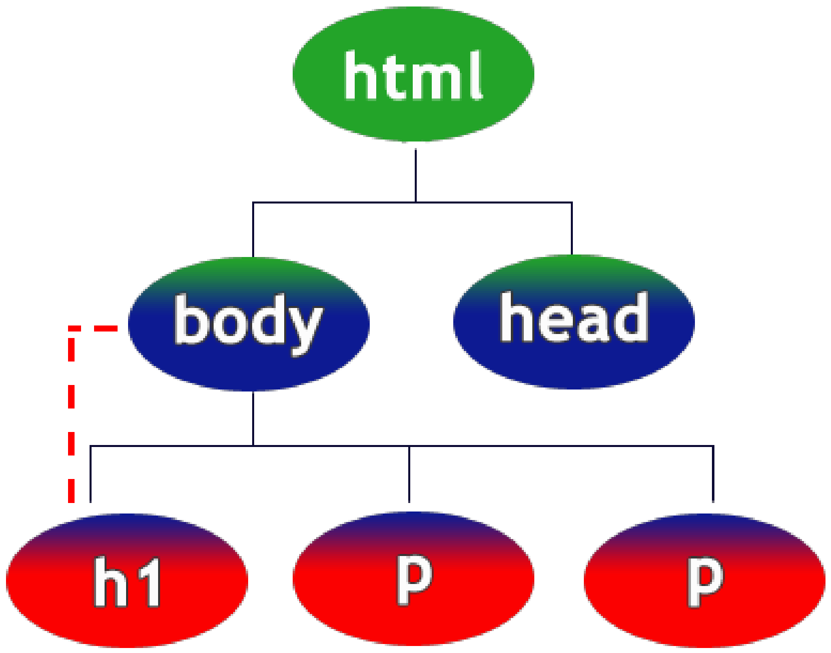
Ancestor/descendant relationship



## HTML

### Document Object Model (DOM)

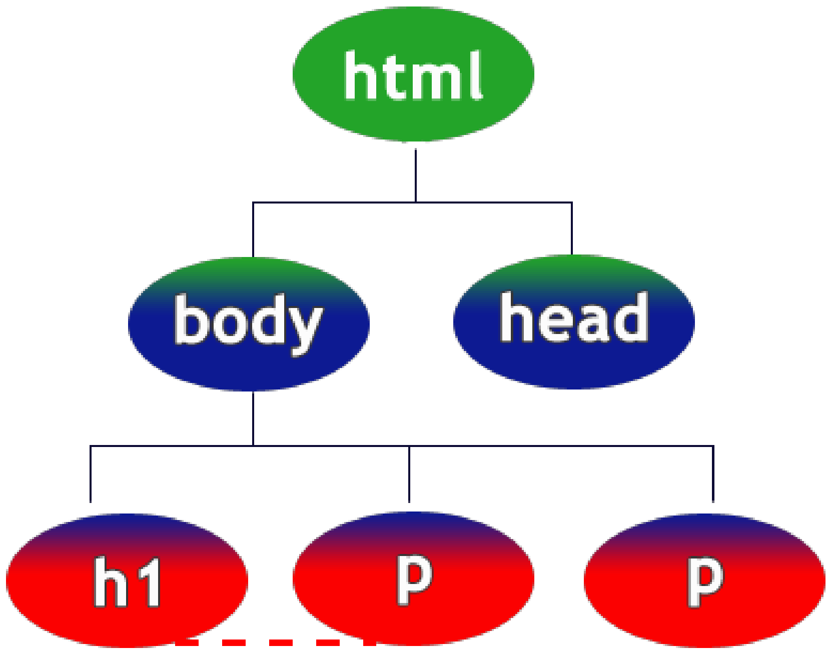
Parent/child relationship



## HTML

### Document Object Model (DOM)

Sibling relationship



## HTML

### Document Object Model (DOM)

**HTML**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict...">
<html xmlns="http://www.w3.org/1999/xhtml">
   <head>
      <title>My Page</title>
   </head>
   <body>
      <p>
         This is my content.
      </p>
   </body>
</html>
```

Often times, designers will use white-space and indentation to help visualize the relationship between elements in the HTML source code. In the above example, you can see that the <head> and <body> tags are indented at the same level, indicating that they are siblings in the DOM hierarchy.

## CSS

### Descendant Selector

E F - Matches any F element that is a **descendant** of an E element

#### CSS

```css
body em { color:red; }
```

#### HTML

```html
<body>
   <p>
      This is my <em>emphasized</em> content.
   </p>
</body>
```

#### Browser

**This is my *emphasized* content.**

Using the descendant selector, we can target all descendant elements of a specific type.

## CSS

### Descendant Selector

E F - Matches any F element that is a **descendant** of an E element

CSS

```
body * { color:red; }
```

HTML

```
<body>
   <p>
      This is my <em>emphasized</em> content.
   </p>
</body>
```

Browser

**This is my *emphasized* content.**

We can combine the descendant selector with the universal selector to target **all** of an elements descendants.

## CSS

### Descendant Selector

E F - Matches any F element that is a **descendant** of an E element

CSS

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

HTML

```
<body>
   <p>
      This is my <em>emphasized</em> content.
   </p>
</body>
```

What color will the text in the <em> tags be...?

## CSS

### Inheritance

- Some CSS properties are passed on from parents to children...
    - color
    - text-align
- Others are not...
    - background-color

## CSS

### Forced Inheritance

Any CSS property can be inherited from its parent if you want it to.

CSS

```css
p { background-color:inherit; }
```

Even though background-color isn't normally inherited, you can tell an element to inherit its parent's background-color by using the above CSS rull.

## CSS

### Descendant Selector

E F - Matches any F element that is a **descendant** of an E element

CSS

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

HTML

```
<body>
   <p>
      This is my <em>emphasized</em> content.
   </p>
</body>
```

Browser

**This is my *emphasized* content.**

Because of inheritance, the <em> element would have a red font as it would inherit it from its parent, the <p> element.

## CSS

### Child Selector

E > F - Matches any F element that is a **child** of an E element (has an E element as its parent)

#### CSS

```
p > em { color:red; }
body > em { color:blue; }
```

#### HTML

```
<body>
   <p>
      This is my <em>emphasized</em> content.
   </p>
</body>
```

#### Browser

**This is my *emphasized* content.**

It is important to note the difference between the descendant selector and the child selector. The child selector will only impact an element's children. The descendant selector will impact an element's children, its children's children, and so on.

## CSS

### Sibling Selector

E + F - Matches any F element that is **immediately** preceded by a sibling E element

**CSS**

```
h1 + p { color:red; }
```

**HTML**

```
<body>
   <h1>Page Title</h1>
   <p>This is some content.</p>
   <p>This is more content.</p>
</body>
```

**Browser**

**Page Title**

**This is some content.**

**This is more content.**

Please note that the sibling selector will only effect the next immediate sibling. It will not impact all siblings! That said, you can target multiple siblings by using multiple sibling selectors. For example,

**CSS**

```
h1 + p + p { color:red; }
```

The selector above would target the second <p> element after the <h1> element.

## CSS

### Mixing Selectors

Any of the selectors that we have discussed can be mixed and matched to achieve greater specificity. For example, if I only wanted to change the display properties of <strong> elements within a <p> element who had a class attribute of 'lyrics', I could use the following:

**CSS**

```
p.lyrics strong { color:red; }
```

**HTML**

```
<body>
 <h1>Song Title</h1>
 <p>
   Here are the lyrics to
   <strong>one of my favorite songs.</strong>
 </p>
 <p class="lyrics">
   I'm seein' <strong>red</strong>...
 </p>
</body>
```

**Browser**

**Song Title**

**Here are the lyrics to one of my favorite songs.**

**I'm seein' red...**

## CSS

### Mixing Selectors

Likewise, if I only wanted to change the display properties of a <p> element which was immediately preceeded by a <h1> element who had an id attribute of 'pageTitle', I could use the following:

**CSS**

```
h1#pageTitle + p { color:red; }
```

**HTML**

```
<body>
   <h1 id="pageTitle">Page Title</h1>
   <p>This is some content.</p>
   <h1>An Article</h1>
   <p>Article content.</p>
   <p>More article content.</p>
</body>
```

**Browser**

**Page Title**

**This is some content.**

**An Article**

**Article content.**

**More article content.**

## CSS

### Selectors

Consider the following HTML and CSS snippets:

**HTML**

```html
<body>
   <p id="article1" class="myContent">
      This is my content
   </p>
</body>
```

**CSS**

```css
     body p { background-color:blue;  }
 p#article1 { background-color:red;   }
p.myContent { background-color:green; }
```

What color would the paragraph be...

## CSS

### Not all CSS selectors are equal...

In the instance that multiple CSS selectors apply to a given element, there are two variables that determine which rules take priority:

- Specificity
- Source Order

## CSS

### Specificity

- Element selector - least specific

CSS

```css
/* Applies to all paragraph elements.
   Not very specific */

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

## CSS

### Specificity

- Class selector - more specific

CSS

```css
/* Applies to all elements with a class attribute of 'bookTitle'.
   Can apply to lots of elements - Not very specific */

.bookTitle
{
   background-color:blue;
}
```

## CSS

### Specificity

- ID selector - even more specific

CSS

```
/* Applies to a single element with an id attribute of 'pageHeading'.
   Can apply to one element - Specific! */

#pageHeading
{
   background-color:green;
}
```

## CSS

### Specificity

- Combined selectors - even more specific

CSS

```css
/* Very specific!! Applies to a strong element with the
   class attribute of 'keyword' that is a descendent
   of a paragraph with an id attribute of 'pageHeading'
   that is a descendent of the body element which
   is the immediate sibling of the head element which
   is a child of the html element.*/

html>head+body p#pageHeading strong.keyword
{
   background-color:green;
}
```

## CSS

### Calculating specificity

By examining the CSS selectors in question, browsers will determine which has greater specificity by evaluating the following...

| # of ID selectors | # of class selectors | # of element selectors |
|---|---|---|
| a | b | c |

... and then combining those values into a number by concatenating them with one another.

abc

## CSS

### Calculating specificity

CSS

```css
body p { background-color:blue; }
```

| # of ID selectors | # of class selectors | # of element selectors |
|---|---|---|
| 0 | 0 | 2 |

In the example above, there are zero ID selectors, zero class selectors, and two element selectors. This leaves us with a specificity value of:

002

## CSS

### Calculating specificity

CSS

```
p#article1 { background-color:red; }
```

| # of ID selectors | # of class selectors | # of element selectors |
|---|---|---|
| 1 | 0 | 1 |

In the example above, there is one ID selector, zero class selectors, and one element selector. This leaves us with a specificity value of:

101

## CSS

### Calculating specificity

CSS

```
p.myContent { background-color:green; }
```

| # of ID selectors | # of class selectors | # of element selectors |
|---|---|---|
| 0 | 1 | 1 |

In the example above, there is zero ID selectors, one class selector, and one element selector. This leaves us with a specificity value of:

011

## CSS

### Calculating specificity

#### HTML

```
<body>
   <p id="article1" class="myContent">
      This is my content
   </p>
</body>
```

#### CSS

```
     body p { background-color:blue;  }  /*** 002 ***/
 p#article1 { background-color:red;    }  /*** 101 ***/
p.myContent { background-color:green; }  /*** 011 ***/
```

Based on specificity, the second CSS selector would take priority and the output would be red.

#### Browser

**This is my content**

## CSS

### Style Attribute - King of Specificity

#### HTML

```
<body>
   <p id="article1" class="myContent" style="background-color:yellow;" >
      This is my content
   </p>
</body>
```

#### CSS

```
     body p { background-color:blue;  }  /*** 002 ***/
 p#article1 { background-color:red;   }  /*** 101 ***/
p.myContent { background-color:green; }  /*** 011 ***/
```

Because the style attribute has greater specificity than any other defined CSS rule, the paragraph element would have a yellow background.

## CSS

### One word to rule them all...

**HTML**

```
<body>
   <p id="article1" class="myContent" style="background-color:yellow;" >
      This is my content
   </p>
</body>
```

**CSS**

```
     body p { background-color:blue !important;  }  /*** ~~002~~ 1002 ***/
 p#article1 { background-color:red;              }  /*** 101 ***/
p.myContent { background-color:green;            }  /*** 011 ***/
```

When the keyword '!important' is added after a CSS property value (before the semicolon), a '1' is added to the front of the specificity value.

Because the '!important' keyword was used on a specific rule, the specificity of the rule is elevated so that it overwrites any other CSS rules or style attributes. In the above example, the paragraph element would have a blue background.

## CSS

### One word to rule them all...

The **!important** keyword should rarely be used!
When it is used, you MUST include comments explaining why it was used.

The !important keyword breaks the natural ordering of specificity. If you don't use it carefully (and commenting reasons for using it), odds are you'll forget where you used it and why. When you modify your CSS file in the future, you'll likely find yourself banging your head on the table when your other CSS rules aren't being applied for no apparent reason, having forgot that you used the !important keyword.

## CSS

### Dueling selectors and source order

HTML

```
<body>
   <p class="intro blogArticle">
      This is my content
   </p>
</body>
```

CSS - styles.css

```
p.blogArticle { background-color:red; }
   .
   .
   .
p.intro { background-color:green; }
```

If conflicting CSS rules exist in a single file and both rules have the same specificity value, the rule which comes last in the file is the one that takes priority.

The paragraph element would have a green background because the second rule came later in the file, overwriting the first.

## CSS

### Dueling selectors and source order

**HTML**

```
<head>
<title>Page Title</title>
<link rel="stylesheet" type="text/css" href="style1.css" />
<link rel="stylesheet" type="text/css" href="style2.css" />
</head>
<body>
   <p class="intro blogArticle"> This is my content </p>
</body>
```

**CSS - style1.css**

```
p.blogArticle { background-color:red; }
```

**CSS - style2.css**

```
p.intro { background-color:blue; }
```

In a case where you have more than one CSS file for a given HTML page, conflicting CSS rules exist in one or more of the files, and all rules have the same specificity value, the rule which comes last in the last specified file is the one that takes priority.

The paragraph element would have a blue background because the file, 'style2.css' was specified in a <link> element after the 'style1.css' file.

## CSS

### Dueling selectors and source order

#### HTML

```
<head>
<title>Page Title</title>
<link rel="stylesheet" type="text/css" href="style2.css" />
<link rel="stylesheet" type="text/css" href="style1.css" />
</head>
<body>
   <p class="intro blogArticle"> This is my content </p>
</body>
```

#### CSS - style1.css

```
p.blogArticle { background-color:red; }
```

#### CSS - style2.css

```
p.intro { background-color:blue; }
```

The paragraph element would have a red background because the file, 'style1.css' was specified in a <link> element after the 'style2.css' file.

## CSS

### Source order and the style attribute

HTML

```
<body>
   <p style="background-color:yellow;background-color:black;">
      This is my content
   </p>
</body>
```

In the case of duplicate CSS rules in the *style* attribute of a given element, the last rule would take priority.