CSS3 – Selectors, Inheritance, Cascading, Specificity,
Tag Selectors: Page-Wide Styling

- A tag selector redefines how a browser displays a particular tag.

- Applies to every occurrence of an HTML tag on a web page.

```css
h2 {
    font-family: "Century Gothic";
    color: #ff0000;
    margin-bottom: 0;
}
```
Class Selectors: Pinpoint Control

- Used when you want to give one or more elements a look that’s different from related tags on a page.

- Created by giving it a name and then applying it to just the HTML tags you wish to format.
  - All class selector names must begin with a full stop.
  - CSS permits only letters, numbers, hyphens, and underscores in class names.
  - After the full stop, the name must always start with a letter. E.g., `.9lives` isn’t a valid class name, but `.crazy8` is.
  - Class names are case sensitive. E.g., CSS considers `.SIDEBAR` and `.sidebar` two different classes.
Class Selectors: Pinpoint Control

- Apart from the name, you create class styles exactly like tag styles.

- After the class name, simply slap on a declaration block containing all of the styling you desire:

  ```
  .special {
      color: #FF0000;
      font-family: "Monotype Corsiva";
  }
  ```

  To indicate where you want to apply that formatting you add a class attribute to the HTML tag you wish to style.

  `<p class="special">`
ID Selectors: Specific Page Elements

- CSS reserves the ID selector for *identifying* a unique part of a page, like a banner, navigation bar, or the main content area.

- Just like a class selector, you create an ID by giving it a name in CSS, and then you apply it by adding the ID to your page’s HTML code.

  ```css
  #banner {
    background: #CC0000;
    height: 300px;
    width: 720px;
  }
  
  <p id="banner">
  ```
ID Selectors: Specific Page Elements

- ID selectors have some specific uses in JavaScript-based or very lengthy web pages. Otherwise, compelling reasons to use IDs over classes are few.

- When deciding whether to use a class or an ID, follow these rules of thumb:
  - To use a style several times on a page, you must use classes. E.g., when you have more than one photo on your page, use a class selector to apply styling - like a border - to each `<img>` tag you wish to style.
  - Use IDs to identify sections that occur only once per page like a sidebar or footer.
  - Consider using an ID selector to sidestep style conflicts, since web browsers give ID selectors priority over class selectors. E.g., when a browser encounters two styles that apply to the same tag but specify different background colours, the ID’s background colour wins.
Group Selectors: Styling Groups of Tags

- A quick way to apply the same formatting to several different elements - simply create a list of selectors separated by commas.

- So to style all of the heading tags with the same colour, you can create the following rule:
  ```css
  h1, h2, h3, h4, h5, h6 { color: #F1CD33; }
  ```

- You can use any valid selector (or combination of selector types) in a group selector.
  ```css
  h1, p,.copyright, #banner { color: #F1CD33; }
  ```
The Universal Selector (Asterisk *)

- An asterisk (*) is universal selector shorthand for selecting every single tag.

- E.g., say you want all the tags on your page to appear in bold type, your group selector might look something like this:

  ```
  a, p, img, h1, h2, h3, h4, h5 ...yadda yadda... { font-weight: bold; }
  ```

- The asterisk, however, is a much shorter way to tell CSS to select all HTML tags on the page:

  ```
  * { font-weight: bold; }
  ```
The Universal Selector (Asterisk)

- You can even use the universal selector as part of a descendant selector, so you can apply a style to all of the tags that descend from a particular page element.

- For example, `#banner *` selects every tag inside the page element to which you’ve applied the `#banner` ID.
Styling Tags Within Tags – Descendent Selectors

- You use descendent selectors to format a whole bunch of tags in a similar manner (just like tag selectors), but only when they’re in a particular part of a web page.

- It’s like saying, “Hey you <a> tags in the navigation bar, listen up. I’ve got some formatting for you. All you other <a> tags, just move along; there’s nothing to see here.”

- Descendent selectors let you format a tag based on its relationship to other tags
Building Descendent Selectors

- Descendent selectors let you take advantage of the DOM (HTML family tree) by formatting tags differently when they appear inside certain other tags or styles.

- For example, say you have an `<h1>` tag on your web page, and you want to emphasize a word within that heading with the `<strong>` tag.

- The trouble is, most browsers display both heading tags and the `<strong>` tag in bold, so anyone viewing the page can’t see any difference between the emphasised word and the other words in the headline.
Creating a tag selector to change the `<strong>` tag’s colour and make it stand out from the headline isn’t much of a solution: You end up changing the colour of every `<strong>` tag on the page, like it or not.

A descendant selector lets you do what you really want - change the colour of the `<strong>` tag only when it appears inside of a `<h1>` tag.

The solution to the `<h1>` and `<strong>` dilemma looks like this:

```
h1 strong { color: red; }
```
Building Descendent Selectors

- Descendent selectors style elements that are nested inside other elements, following the exact same pattern of ancestors and descendants as the tags in the HTML family tree.

- You create a descendent selector by tacking together selectors according to the part of the family tree you want to format, with the most senior ancestor on the left and the actual tag you’re targeting on the far right.
Building Descendent Selectors

- You’re not limited to just tag selectors, either. You can build complex descendent selectors combining different types of selectors.

- E.g., suppose you want your links to appear in yellow only when they’re in introductory paragraphs (which you’ve designated with a class style named `intro`). The following selector does the trick:
  ```css
  p.intro a { color: yellow; }
  ```

- If you add a space, you get a different effect:
  ```css
  p .intro a { color: yellow; }
  ```
  This slight variation selects an `<a>` tag inside any tag styled with the `.intro` class, which is a descendent of a `<p>` tag.
Other Selectors

- There are also Pseudo-Classes and Pseudo-Elements covered in lecture 35
- And Attribute Selectors covered in lecture 34.
Similar to the descendendent selectors described earlier in this chapter, CSS lets you format the children of another tag with a child selector.

The child selector uses an angle bracket (>) to indicate the relationship between the two elements.

E.g., the selector body > h1 selects any <h1> tag that’s a child of the <body> tag.

Unlike a descendendent selector, which applies to all descendants of a tag (children, grandchildren, and so on), the child selector lets you specify which child of which parent you mean.
Advanced Selectors – Child Selectors

- Here there are two `<h2>` tags. Using a plain old descendent selector - `body h2` - selects both `<h2>` tags. Even though both `<h2>` tags are inside the `<body>` tag, only the second one is a child of the `<body>` tag.

- The first `<h2>` is directly inside a `<div>` tag, so its parent is `<div>`.
  Since the two `<h2>` tags have different parents, you can use a child selector to get at them individually.

- To select only the second `<h2>` tag, your child selector looks like this: `body > h2`. If you want the first `<h2>` tag, then you must use this child selector instead: `div > h2`. 
Advanced Selectors – Adjacent Siblings

- Parent-child relationships aren’t the only ones in the HTML family tree.

- Sometimes you need to select a tag based not on its parent tag but on its surrounding siblings - the tags that share a common parent.

- A tag that appears immediately after another tag in HTML is called an adjacent sibling.
Advanced Selectors – Adjacent Siblings

- Here the `<div>` tag is the adjacent sibling of the `<h1>` tag, the `<p>` tag is the adjacent sibling of the `<h2>` tag, and so on.

- The adjacent sibling selector uses a plus sign (+) to join one element to the next.

- So to select every paragraph following each `<h2>` tag, use this selector:
  \[ h2 + p \]

- The last element in the selector (p, in this case) is what gets the formatting, but only when it’s directly after its sibling `<h2>`.
Inheritance

- Inheritance is the process by which some CSS properties applied to one tag are passed on to nested tags.

- For example, a `<p>` tag is always nested inside of the `<body>` tag, so properties applied to the `<body>` tag get inherited by the `<p>` tag.

- Say you created a CSS tag style for the `<body>` tag that sets the `color` property to a dark red. Tags that are descendants of the `<body>` tag - that is, the ones inside the `<body>` tag - will inherit that `color` property.

- That means that any text in those tags - `<h1>`, `<h2>`, `<p>`, whatever - will appear in that same dark red colour.
Inheritance

- Inheritance doesn’t just apply to tag styles. It works with any type of style.

- so when you apply a class style to a tag, any tags inside that tag inherit properties from the styled tag.

- Same holds true for ID styles, descendent selectors, and the other types of styles
How Inheritance Streamlines Style Sheets

- You can use inheritance to your advantage to streamline your style sheets.

- Say you want all the text on a page to use the same font. Instead of creating styles for each tag, simply create a tag style for the `<body>` tag. (Or create a class style and apply it to the `<body>` tag.)

- In the style, specify the font you wish to use, and all of the tags on the page inherit the font:

  
  ```
  body { font-family: Arial, Helvetica, sans-serif; }
  ```
How Inheritance Streamlines Style Sheets

- You can also use inheritance to apply style properties to a section of a page.

- E.g., you may use the `<div>` tag to define an area of a page like a banner, sidebar, or footer. By applying a style to a `<div>` tag, you can specify particular CSS properties for all of the tags inside just that section of the page.

- If you want all the text in a sidebar to be the same colour, you’d create a style setting the `color` property, and then apply it to the `<div>`.

- Any `<p>`, `<h1>`, or other tags inside the `<div>` inherit the same font colour.
The Limits of Inheritance

- Many CSS properties don’t pass down to descendent tags at all.

- For example, the `border` property isn’t inherited, and with good reason. If it were, then every tag inside an element with the `border` property would also have a border around it.

- E.g., if you added a border to the `<body>` tag, then every bulleted list would also have a box around it, and each bulleted item in the list would also have a border
Examples of times when inheritance doesn’t strictly apply

- As a general rule, properties that affect the placement of elements on the page or the margins, background colours, and borders of elements aren’t inherited.

- Web browsers use their own default styles to format various tags: Headings are big and bold, links are blue, and so on. When you define a font-size for the text on a page and apply it to the `<body>` tag, headings still appear larger than paragraphs, and `<h1>` tags are still larger than `<h2>` tags. It’s the same when you apply a font colour to the `<body>`; the links on the page still appear in good old-fashioned, web-browser blue.
Examples of times when inheritance doesn’t strictly apply

- When styles conflict, the more specific style wins out.

- In other words, when you’ve specifically applied CSS properties to an element—like specifying the font size for an unordered list—and those properties conflict with any inherited properties—like a font-size set for the `<body>` tag—the browser uses the font size applied to the `<ul>` tag.
Managing Multiple Styles: The Cascade

- The *cascade* is a set of rules for determining which style properties get applied to an element.

- It specifies how a web browser should handle multiple styles that apply to the same tag and what to do when CSS properties conflict.
Managing Multiple Styles: The Cascade

Style conflicts happen in two cases:

1. Through inheritance when the same property with different values is inherited from multiple ancestors.

2. When one or more styles apply to the same element (maybe a `<p>` tag style in an external style sheet and another `<p>` tag style in an internal style sheet).
Inherited Styles Accumulate

- Since one tag can inherit properties from any ancestor tag - a link, for example, inheriting the same font as its parent `<p>` tag - determining why a particular tag is formatted one way can be a bit tricky.

- Imagine a font family applied to the `<body>` tag, a font size applied to a `<p>` tag, and a font colour applied to an `<a>` tag. Any `<a>` tag inside of a paragraph would inherit the font from the body and the size from the paragraph.

- In other words, the inherited styles combine to form a hybrid style.
Inherited Styles Accumulate

body {
    font-family: Verdana, Arial, Helvetica, sans-serif;
}

p {
    color: #F30;
}

strong {
    font-size: 24px;
}
Inherited Styles Accumulate

- That `<strong>` tag inherits from both of its ancestors, so it inherits the `font-family` property from the body and the `color` property from its parent paragraph.

- In addition, the `<strong>` tag has a bit of CSS applied directly to it - a 24px font size.

- The final appearance of the tag is a combination of all three styles. In other words, the `<strong>` tag appears exactly as if you’d created a style like this:

```
strong {
    font-family: Verdana, Arial, Helvetica, sans-serif;
    color: #F30;
    font-size: 24px;
}
```
Nearest Ancestor Wins

- What happens when inherited CSS properties conflict?

- Think about a page where you’ve set the font colour for both the body and paragraph tags.

- Now imagine that within one paragraph, there’s a `<strong>` tag. Which colour gets applied to text inside the `<strong>` tag? The colour inherited from the body or the paragraph?

- Ladies and gentleman, we have a winner: the paragraph. That’s because the web browser obeys the style that’s closest to the tag in question.
Nearest Ancestor Wins

- In this example, any properties inherited from the `<body>` tag are rather generic. They apply to all tags.

- A style applied to a `<p>` tag, on the other hand, is much more narrowly defined. Its properties apply only to `<p>` tags and the tags inside them.

- In a nutshell, if a tag doesn’t have a specific style applied to it, then, in the case of any conflicts from inherited properties, the nearest ancestor wins.
Nearest Ancestor Wins

- Example 2: if a CSS style defining the colour of text were applied to a `<table>` tag, and another style defining a different text colour were applied to a `<td>` tag inside that table, then tags inside that table cell (`<td>`) such as a paragraph, headline, or unordered list would use the colour from the `<td>` style, since it’s the closest ancestor.
The Directly Applied Style Wins

- Taking the “nearest ancestor” rule to its logical conclusion, there’s one style that always becomes king of the CSS family tree - any style applied directly to a given tag.

- Suppose a font colour is set for the body, paragraph, and strong tags. The paragraph style is more specific than the body style, but the style applied to the `<strong>` tag is more specific than either one. It formats the `<strong>` tags and only the `<strong>` tags, overriding any conflicting properties inherited from the other tags.

- In other words, properties from a style specifically applied to a tag beat out any inherited properties.
The Directly Applied Style Wins

- This rule explains why some inherited properties don’t appear to inherit.

- A link inside a paragraph whose text is red still appears browser-link blue. That’s because browsers have their own predefined style for the `<a>` tag, so an inherited text colour won’t apply.
One Tag, Many Styles

- Inheritance is one way that a tag can be affected by multiple styles. But it’s also possible to have multiple styles apply *directly* to a given tag.

- E.g., say you have an *external* style sheet with a `<p>` tag style and attach it to a page that has an *internal* style sheet that *also* includes a `<p>` tag style. And just to make things really interesting, one of the `<p>` tags on the page has a class style applied to it.

- So for that one tag, three different styles directly format it. Which style - or *styles* - should the browser obey?
It depends. Based on the types of styles and the order in which they’re created, a browser may apply one or more of them at once.

Here are a few situations in which multiple styles can apply to the same tag:

- The tag has both a tag selector and a class style applied to it.
- E.g., a tag style for the `<h2>` tag, a class style named `.leadHeadline` and this HTML:
  
  `<h2 class="leadHeadline">Your Future Revealed!</h2>`

  Both styles apply to this `<h2>` tag.
One Tag, Many Styles

- The same style name appears more than once in the style sheet.
- There could be a group selector, like `.leadHeadline, .secondaryHeadline, .newsHeadline`, and the class style `.leadHeadline` in the same style sheet. Both of these rules define how any element with a class of `leadHeadline` looks.

- A tag has both a class and an ID style applied to it. Maybe it’s an ID named `#banner`, a class named `.news`, and this HTML: `<div id="banner" class="news">`. Properties from both the `banner` and `news` styles apply to this `<div>` tag.
One Tag, Many Styles

- There's more than one style sheet containing the same style name attached to a page.
- The same-named styles can arrive in an external style sheet via @import or link and an internal style sheet.

- There are complex selectors targeting the same tag.
- Common when you use descendent selectors. E.g., say you have a div tag in a page (<div id="mainContent">), and inside the div is a paragraph with a class applied to it: <p class="byline">. The following selectors apply to this paragraph:

```markdown
#mainContent p
#mainContent .byline
p.byline
.byline
```
One Tag, Many Styles

- If more than one style applies to a particular element, then a web browser **combines** the properties of all those styles, *as long as they don’t conflict.*

- E.g., you have a paragraph that lists the name of the web page’s author, including a link to their email address.
  
  ```html
  <p class="byline">Written by <a href="mailto:Myname@gmail.com">My Name</a></p>
  ```

- Meanwhile, the page’s style sheet has three styles that format the link
  ```css
  a {
    color: #6378df;
  }
  p a {
    font-weight: bold;
  }
  .byline a {
    text-decoration: none;
  }
  ```
One Tag, Many Styles

- The first style turns all `<a>` tags powder blue; the second style makes all `<a>` tags that appear inside a `<p>` tag bold; and the third style removes the underline from any links that appear inside an element with the `byline` class applied to it.

- All three styles apply to that very popular `<a>` tag, but since none of the properties are the same, there are no conflicts between the rules.

- The styles combine to make one überstyle containing all three properties, so this particular link appears powder blue, bold, and underline-free.
Specificity: Which Style Wins?

- The previous example is pretty straightforward. But what if the three styles listed on each had a different font specified for the `font-family` property? Which of the three fonts would a web browser pay attention to?

- As you know, the cascade provides a set of rules that helps a web browser sort out any property conflicts; namely, *properties from the most specific style win*.

- But sometimes it’s not clear which style is most specific.
Specificity: Which Style Wins?

- Thankfully, CSS provides a formula for determining a style’s specificity that’s based on a value assigned to the style’s selector - a tag selector, class selector, ID selector, and so on.

- Here’s how the system works:
  - A tag selector is worth 1 point.
  - A class selector is worth 10 points.
  - An ID selector is worth 100 points.
  - An inline style is worth 1,000 points.
Specificity: Which Style Wins?

- The bigger the number, the greater the specificity. So say you create the following three styles:
  - a tag style for the `<img>` tag (specificity = 1)
  - a class style named `.highlight` (specificity = 10)
  - an ID style named `#logo` (specificity = 100)

- Then, say your web page has this HTML:
  ```html
  <img id="logo" class="highlight" src="logo.gif" />
  ```

- If you define the same property - such as the border property - in all three styles, then the value from the ID style (`#logo`) always wins out.
Specificity: Which Style Wins?

- A pseudo-element (like :first-line for example) is treated like a tag selector and is worth 1 point.

- A pseudo-class ( :link, for example) is treated like a class and is worth 10 points.

- Inherited properties don’t have any specificity. So even if a tag inherits properties from a style with a large specificity - like #banner – those properties will always be overridden by a style that directly applies to the tag.
Specificity: Which Style Wins?

- Since descendent selectors are composed of several selectors - *#content p*, or *h2 strong*, for example - the maths gets a bit more complicated. The specificity of a descendent selector is the total value of all of the selectors listed.

<table>
<thead>
<tr>
<th>selector</th>
<th>id</th>
<th>class</th>
<th>tag</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>p</em></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td><em>.byline</em></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td><em>p.byline</em></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td><em>#banner</em></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td><em>#banner p</em></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>101</td>
</tr>
<tr>
<td><em>#banner .byline</em></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>110</td>
</tr>
<tr>
<td><em>a:link</em></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td><em>p:first-line</em></td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td><em>h2 strong</em></td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td><em>#wrapper #content .byline a:hover</em></td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>221</td>
</tr>
</tbody>
</table>

Note: The math involved in calculating specificity is actually a bit more complicated than described here. But this formula works in all but the weirdest cases. To read how web browsers actually calculate specificity visit [www.w3.org/TR/CSS21/cascade.html#specificity](http://www.w3.org/TR/CSS21/cascade.html#specificity).
The Tiebreaker: Last Style Wins

- It’s possible for two styles with conflicting properties to have the same specificity.

- A specificity tie can occur when you have the same selector defined in two locations.

- You may have a `<p>` tag selector defined in an internal style sheet and an external style sheet.

- Or two different styles may simply have equal specificity values.

- In case of a tie, the style **appearing last in the style sheet wins**.
The Tiebreaker: Last Style Wins

- Take the following HTML:
  
  ```html
  <p class="byline">Written by <a class="email" href="mailto:myname@gmail.com">My Name</a></p>
  ```

- In the style sheet for the page containing the above paragraph and link, you have two styles:
  ```css
  p .email { color: blue; }
  .byline a { color: red; }
  ```

- Both styles have a specificity of 11 (10 for a class name and 1 for a tag selector) and both apply to the `<a>` tag. The two styles are tied.
The Tiebreaker: Last Style Wins

- Which colour does the browser use to colour the link?

- Answer: Red, since it’s the second (and last) style in the sheet.

- Now suppose the style sheet looked like this instead:
  ```
  .byline a { color: red; }
  p .email { color: blue; }
  ```

- In this case, the link would be blue. Since `p .email` appears after `.byline a` in the style sheet, its properties win out.
The Tiebreaker: Last Style Wins

- What happens if you’ve got conflicting rules in an external and an internal style sheet? In that case, the placement of your style sheets (within your HTML file) becomes very important.

- If you first add an internal style sheet using the `<style>` tag and then attach an external style sheet farther down in the HTML using the `<link>` tag, then the style from the external style sheet wins.

- In effect, it’s the same principle at work: *The style appearing last wins.*

- The bottom line: Be consistent in how you place external style sheets. It’s best to list any external style sheets first, and then include any internal styles.
You can use this trick when you absolutely, positively want to make sure that a particular property can’t be overridden by a more specific style.

Simply insert `!important` after any property to shield it from specificity-based overrides.
Overruling Specificity

- Consider the two following styles:
  ```css
  #nav a { color: red; }
a { color: teal !important; }
  ```

  Normally, a link inside an element with the ID of `#nav` would be coloured red since the `#nav a` style is much more specific than the `a` tag style.

  However, including `!important` after a property value means that specific property always wins. So in the above example, all links on the page - including those inside an element with the `#nav` id - are teal.
Overruling Specificity

- Note that you apply !important to an individual property, not an entire style.

- Finally, when two styles both have !important applied to the same property, the more specific style’s !important rule wins.