CSS Inheritance and Specificity

The "cascading" part of CSS comes from the cascade of rules that get applied to determine the styling of an element.

Firstly, many properties are **inherited**. For example, if the **color** of an **article** was set to red, by default the **p** elements would also have their text red, as would any **b** elements, say, nested within them.

But not all properties are inherited. If an **article** was given a dashed blue border, the **paragraphs** within it would not themselves be given such a border by default.
CSS Inheritance and Specificity

Sometimes CSS stylings could override inheritance and if we don’t want that, we can specify a value of `inherit` for most properties.

Another value many properties can be assigned is `initial`, representing their unmodified state. For example,

```css
p {
    color: red;
}
b {
    color: initial;
}
```
Given code above, a `b` within `p` would appear in its default color, not red.
CSS Inheritance and Specificity

When two different stylings could apply to a single element, the styling that is applied is set by the **specificity**. Specificity requires calculating a 4-tuple, \((d_1, d_2, d_3, d_4)\), for each element.

- \(d_1 = 1\) if there is an inline style applied (shouldn’t be done!) and \(d_1 = 0\) otherwise.
- \(d_2 = \) the number of id selectors, the #’s used.
- \(d_3 = \) the number of classes and pseudo-classes used.
- \(d_4 = \) the number of elements and pseudo-elements used.

Specificity is ordered **lexicographically** based on the 4-tuple.
CSS Inheritance and Specificity

For some examples:
* { /* (0,0,0,0): select all does not list any elements, classes, ids, etc. */
  color: red;
}

p::selection { /* (0,0,0,2): element and pseudo-element */
  color: red;
}

#setup .important { /* (0,1,1,0): one id, one class */
  color: red;
}

.first_part .important { /* (0,0,2,0): two classes */
  color: blue;
}
CSS Inheritance and Specificity

When two items tie for specificity, the most recent (farthest down on the page) wins out.

The most specific style possible will apply so that stylings for p will override styles for main, for example.
CSS Inheritance and Specificity I

Once upon a time, there were students studying PIC 40A...

And they all lived happily ever after.

with CSS:

```css

i { /* describes the O */
  color: black;
}

b { /* describes b's */
  font-size: 20em;
}

b { /* overrides above b font-size */
  font-size: 2em;
}

do { /* describes the O */
  color: green;
  font-size: 1em;
}

#setup .important{
  color: red;
}

```
CSS Inheritance and Specificity II

22 .first_part .important{
23     color: orange;
24 }
25
26 .first_part {
27     color: blue;
28 }

CSS Inheritance and Specificity

The following output would be generated:

Once upon a time, there were students studying PIC 40A...

And they all lived happily ever after.

By default all `p` elements are given green color and standard font-size.

The class `first_part` comes within the `p` element so we start to see the blue font specified by the `.first_part`.

The `i` element comes within the `important` class so we see the style applied to `i` of black is applied instead of the style specified by `.first_part`. 
The word "time" can be described as either descended from \#setup.important or .first_part .important. The former has specificity (0,1,1,0) > (0,0,2,0) so the red of \#setup .important wins because it has a higher specificity.

b is specified twice but only the style furthest down is considered, hence font-size of double vs twenty times. For the "A", b is more descriptive than p, so the b font size (2 em) wins out.