

# CS193X: Web Programming Fundamentals

Spring 2017

Victoria Kirst  
([vrk@stanford.edu](mailto:vrk@stanford.edu))

# Today's schedule

## Schedule:

- HTML: Background and history
- Complex selectors
- Box model
- Debugging with Chrome Inspector
- **Case study:** Squarespace Layout (will continue into Monday)

# (Forgot to mention: Paths)

`img src`, `a href`, and `link href` can all take either **relative** or **absolute** paths to the resource:

- `<a href="about.html">About</a>`
- ``
- `<link rel="stylesheet" href="css/style.css"/>`

If you are unfamiliar with paths, check out the following:

- [Absolute vs relative paths](#)
- [Unix directories and file paths](#)
- If anything's still unclear, come to [office hours!](#)

# HTML: Background and History

# What HTML elements can I use?

Q: Instead of `<span class="highlight"></span>`, can I create a `<highlight>` element?

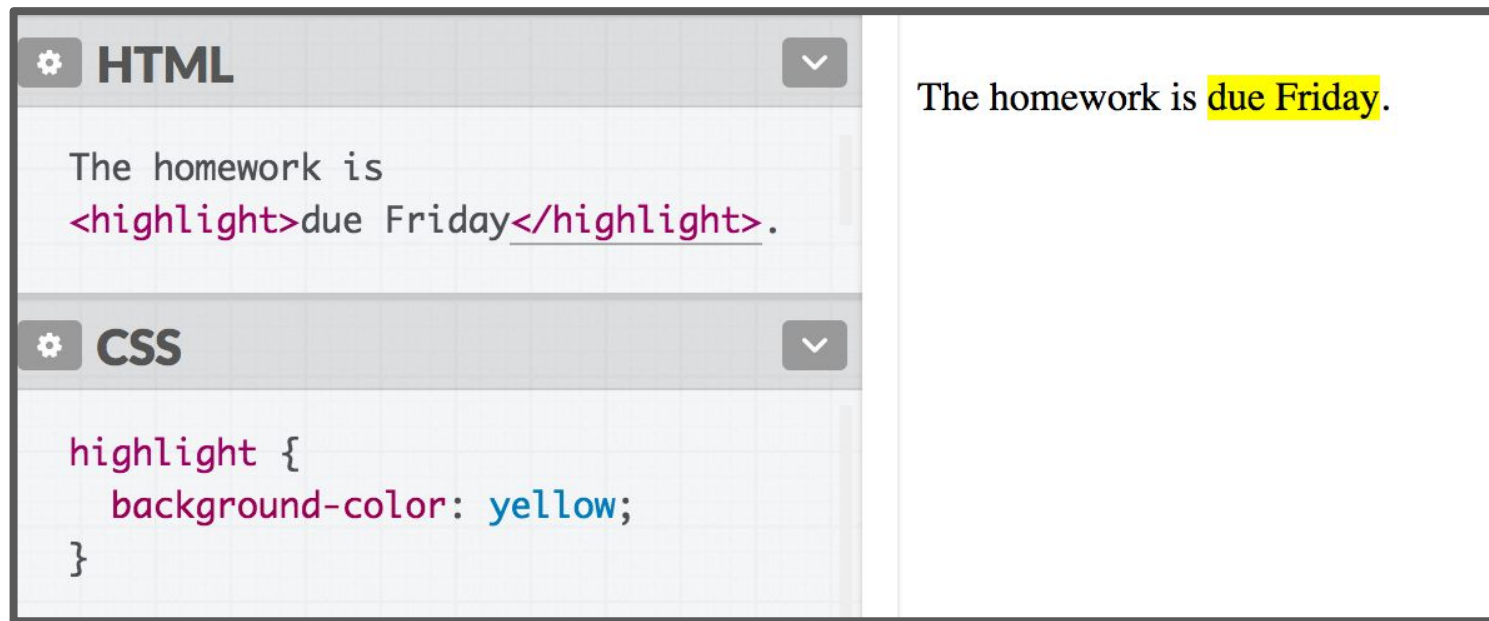
```
<p>
  The homework is
  <highlight>due Friday</highlight>.
</p>
```

```
highlight {
  background-color: yellow;
}
```

Q: Does this even work?

# What HTML elements can I use?

This renders correctly:



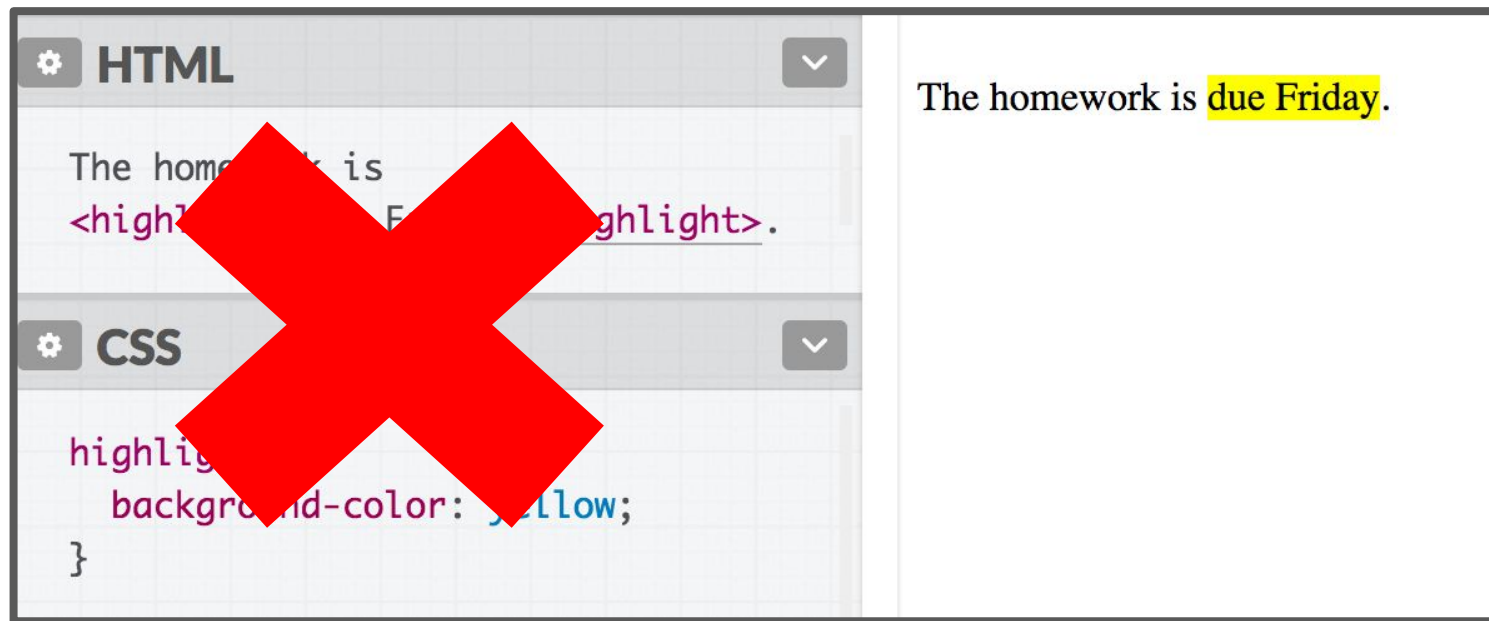
```
HTML
The homework is
<highlight>due Friday</highlight>.

CSS
highlight {
  background-color: yellow;
}
```

The homework is due Friday.

# What HTML elements can I use?

This renders correctly:

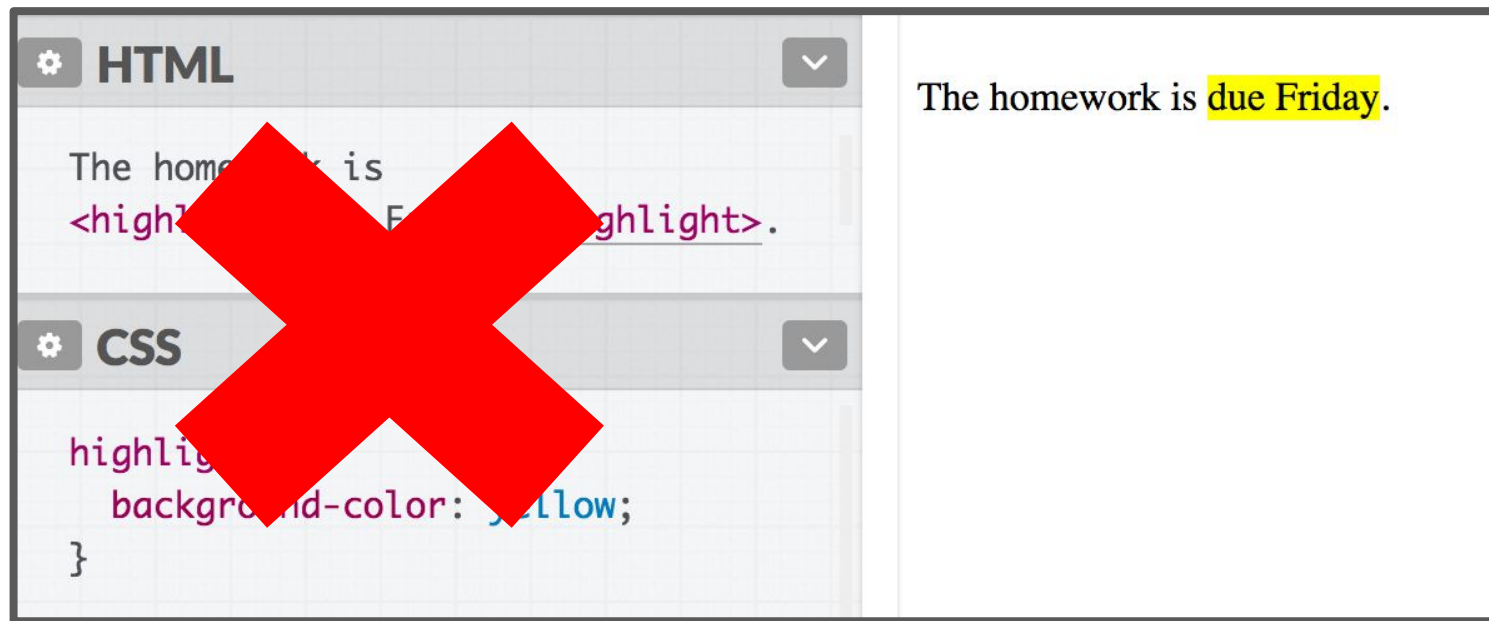


**But you shouldn't do this!**

It is non-standard behavior.

# What HTML elements can I use?

This renders correctly:



**What?!?!?**

**But you shouldn't do this!**  
It is non-standard behavior.



# What?!

- What is "standard" HTML?
- Why does invalid HTML/CSS still work sometimes?
  - If my Java code is wrong, I get a compiler error... If my HTML or CSS is wrong, why don't I get an error?
- Why does it matter that I follow "standard" HTML?

# A very brief history of HTML

# History



**Tim  
Berners-Lee**

- 1989: World Wide Web created  
(WWW: web pages and the protocol in which they are served HTTP/HTTPS)
- 1994: World Wide Web Consortium created
  - "**W3C**": Goal to maintain and develop standards about how the web should work
  - Oversees several languages:
    - HTML, CSS, DOM, XML, etc
- 1997: "HTML4" published
  - The first major stable version of HTML

# Degrading gracefully

The W3C HTML spec lists several [design principles](#), and one is degrading gracefully:



"An escalator  
can never  
break: it can  
only become  
stairs"

This is why browsers do a **best-effort** to render non-standard ("invalid") HTML and CSS.

# Best-effort rendering



The homework is `<highlight>due Friday</highlight>`

highlight {  
 background-color: yellow;  
}

The homework is **due Friday.**

**Don't do this!**

**Don't do this!**

**Don't do this!**

It's also why `<highlight>` "works", even though it's Invalid HTML.

# Why not enforce strict HTML?

It's super weird that:

- Browsers don't fail when given invalid HTML / CSS
- Browsers not only don't fail, but they render invalid HTML/CSS seemingly "correctly"

**Q: Why doesn't the browser reject poorly written HTML/CSS?**

# Why not enforce strict HTML?

It's super weird that:

- Browsers don't fail when given invalid HTML / CSS
- Browsers not only don't fail, but they render invalid HTML/CSS seemingly "correctly"

Q: Why doesn't the browser reject poorly written HTML/CSS?

**A: There was a (failed) attempt to enforce this, but it was too late: the Internet grew too big!**

# The nerdy, mostly\* accurate backstory for HTML today

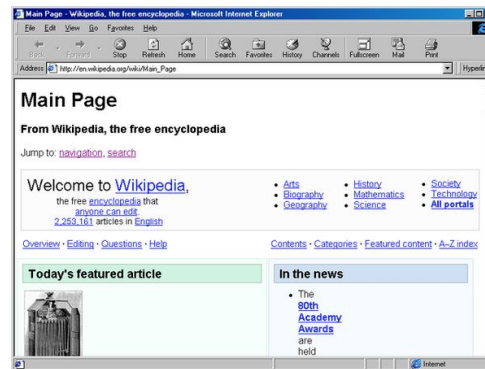
\*I would be more accurate, but it's hard to get valid sources online... so I'm going off of what I can + the lore I've heard while working on a browser.



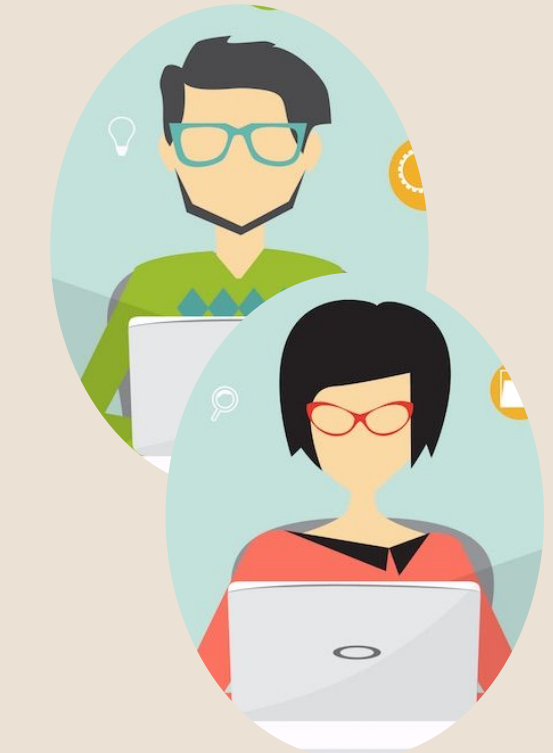
# State of the world, 1997:



Standards say  
one thing,



Browsers do  
another thing,



Developers write  
weird, non-standard  
code.

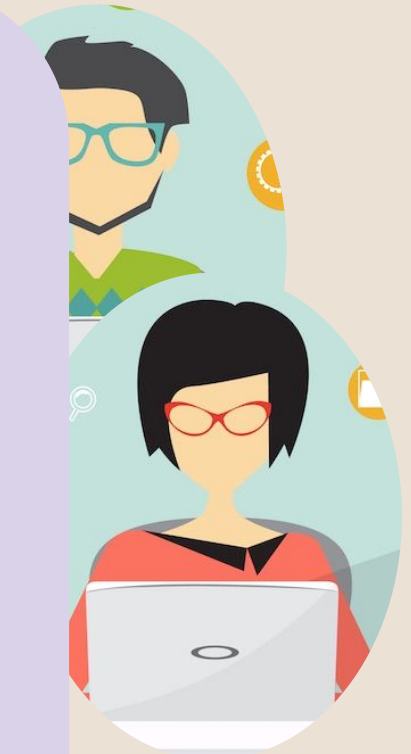
# State of the world, 1997:



Standard  
one thing,

**In 1997, things are  
kind of a mess!**

another thing,



Developers write  
weird, non-standard  
code.

# 2000ish:



**W3C**

Oops, that "degrading gracefully" thing was a mistake.

Browsers, stop rendering pages that have invalid HTML.

**(This was the proposal of XHTML 1.1)**

# 2000ish: (not totally accurate)



W3C

**That would  
break the  
internet!**

**What?!?!**

Sure  
whatever



**We can't  
do that!**



# 2004: WHATWG formed



**Let's burn everything and  
start from scratch with  
XHTML 1.1**

(break approx. 64 million websites)



**WHATWG**



**Let's work on HTML5**  
(an imperfect but realistic standard)

# Fast forward 2017?!



- W3C gave up XHTML 1.1 in 2007
- W3C and WHATWG are mostly friends (I think), though they are still separate entities
- Can still find some snarky quotes on [WHATWG website](#)

# "HTML5" vs HTML

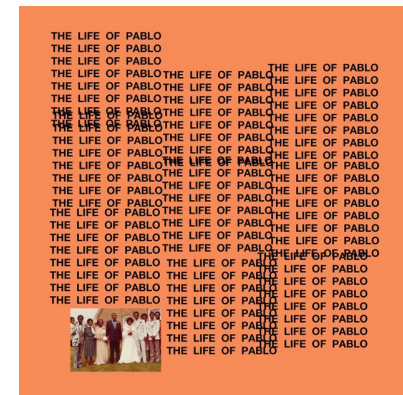
W3C maintains [HTML5](#):

- More stable version of WHATWG's HTML
- Usually copies what WHATWG does after the dust settles



WHATWG maintains [HTML: The Living Standard](#)

- No number, no versions
- Updated frequently and being updated today!
- Most browsers implement WHATWG
- This is why I don't say "HTML5"



# What you need to know

**Q: What HTML elements can I choose from?**

- Check [MDN's list of HTML tags](#)

**Q: How do I know if an HTML tag (or CSS property, or JS feature) is implemented on all browsers?**

- Check [caniuse.com](#)

**Q: Why shouldn't I use non-standard HTML/CSS/JavaScript, even if it works in every browser?**



# What you need to know

**Q: What HTML elements can I choose from?**

- Check [MDN's list of HTML tags](#)

**Q: How do I know if an HTML tag (or CSS property, or JS feature) is implemented on all browsers?**

- Check [caniuse.com](#)

**Q: Why shouldn't I use non-standard HTML/CSS/JavaScript, even if it works in every browser?**

- Because it won't be guaranteed to work in the future
- Because it won't be guaranteed to work on all "user agents" (not just browsers)

# What you need to know

## Q: Wouldn't it be super useful to create custom elements?

- Yes! There is a [spec for this](#) currently under development.
  - (Note that custom elements are not really meant for our example; custom elements are meant for defining custom behavior and not just style. For defining style, CSS classes/ids are still most appropriate.)

Back to writing  
code!

# CSS Selectors: Classes and Ids

# Classes and ids

There are 3 basic types of CSS selectors:

Element selector (this is the one we've been using)	<b>p</b>	All <b>&lt;p&gt;</b> elements
❖ ID selector ❖	<b>#abc</b>	element with <b>id="abc"</b>
❖ Class selector ❖	<b>.abc</b>	elements with <b>class="abc"</b>

```
<h1 id="title">Homework</h1>
<em class="hw">HW0</em> is due Friday.<br/>
<em class="hw">HW1</em> goes out Monday.<br/>
<em>All homework due at 11:59pm.</em>
```

Other selectors

# *element.className*

Syntax	Example	Example described
<i>element.className</i>	<code>p.abc</code>	<code>&lt;p&gt;</code> elements with abc class

```
HTML
<h1 class="hw">Homework 0</h1>
<p class="hw">Due Fri</p>
<p class="hw">Late cutoff Sun</p>
<h1>Lectures</h1>
<p>Apr 3: Syllabus</p>
<p>Apr 5: HTML+CSS</p>

CSS
p.hw {
  color: green;
}
```

## Homework 0

Due Fri

Late cutoff Sun

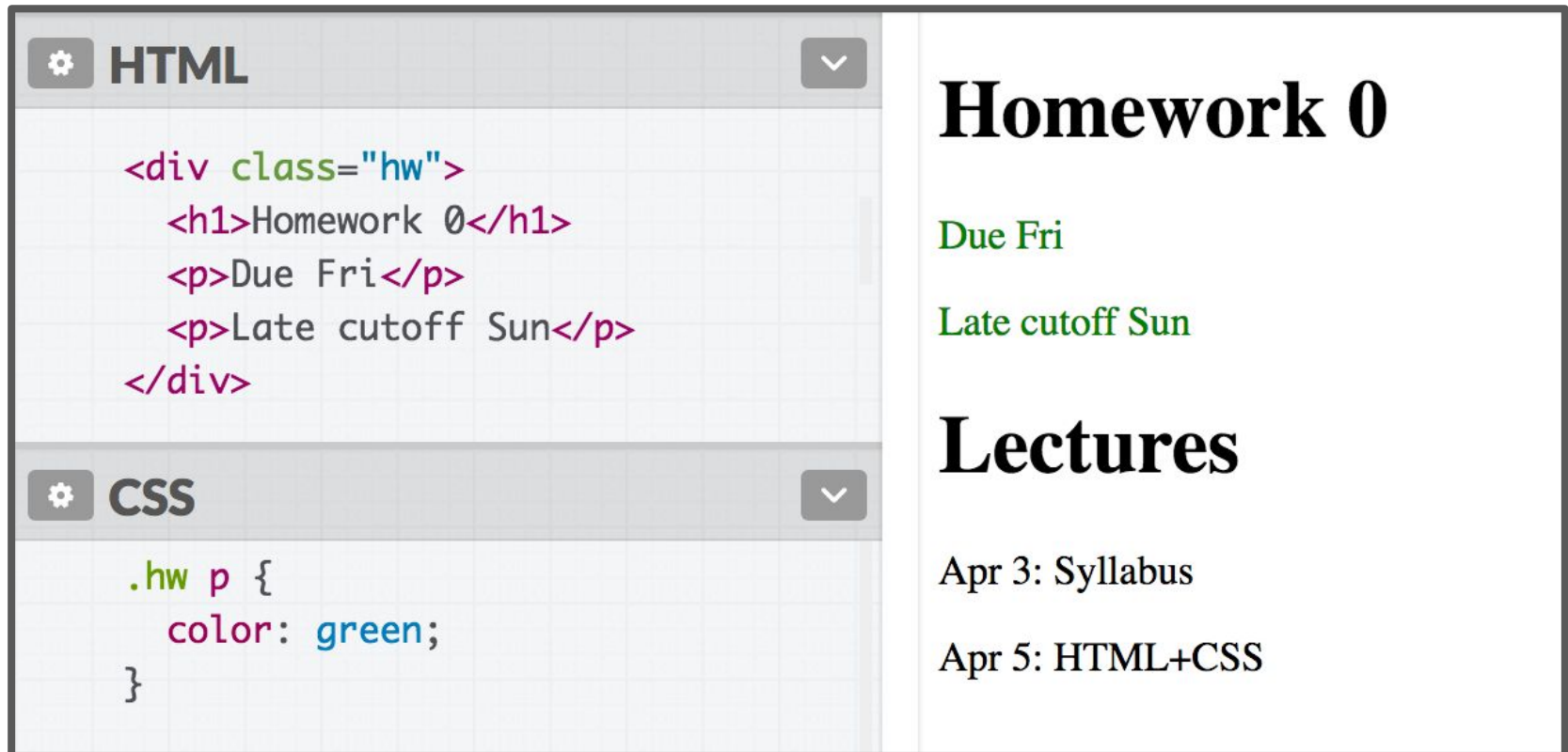
## Lectures

Apr 3: Syllabus

Apr 5: HTML+CSS

# Descendent selector

Syntax	Example	Example described
<i>selector selector</i>	<code>div strong</code>	<code>&lt;strong&gt;</code> elements that are descendants of a <code>&lt;div&gt;</code>



The screenshot shows a code editor with two panels: HTML and CSS. The HTML panel contains the following code:

```
<div class="hw">
  <h1>Homework 0</h1>
  <p>Due Fri</p>
  <p>Late cutoff Sun</p>
</div>
```

The CSS panel contains the following code:

```
.hw p {
  color: green;
}
```

The rendered output on the right side of the editor shows the following content:

# Homework 0

Due Fri

Late cutoff Sun

## Lectures

Apr 3: Syllabus

Apr 5: HTML+CSS



# Descendent selector

Syntax	Example	Example described
<i>selector selector</i>	<b>div strong</b>	<b>&lt;strong&gt;</b> elements that are descendants of a <b>&lt;div&gt;</b>

**Note:** The element does not have to be a direct child. The descendent may be nested many layers in.

The screenshot shows a code editor with two panels: HTML and CSS. The HTML panel contains the following code:

```
<div class="hw">
  <div>
    <p>
      HW0: <strong>Due Friday</strong>
    </p>
  </div>
  HW1 out <strong>Monday</strong>
</div>
```

The CSS panel contains the following code:

```
.hw strong {
  color: red;
}
```

The preview on the right shows the rendered output:

HW0: **Due Friday**  
HW1 out **Monday**

# Descendent selector

Syntax	Example	Example described
<i>selector selector</i>	<code>div strong</code>	<code>&lt;strong&gt;</code> elements that are descendants of a <code>&lt;div&gt;</code>

## Discouraged:

```
<h1 class="hw">Homework 0</h1>  
<p class="hw">Due Fri</p>  
<p class="hw">Late cutoff Sun</p>
```

vs

## Preferred:

```
<div class="hw">  
  <h1>Homework 0</h1>  
  <p>Due Fri</p>  
  <p>Late cutoff Sun</p>  
</div>
```

Instead of applying a class to several adjacent elements, wrap the group in a `<div>` container and style the contents via descendent selectors.

# selector, selector (comma)

Syntax	Example	Example described
<i>selector, selector</i>	<b>h2, div</b>	<h2> elements and <div>s

The screenshot shows a web browser interface. On the left, there are two panels: 'HTML' and 'CSS'. The HTML panel contains the following code:

```
<h1>Course Info</h1>
<h2>Lectures</h2>
<p>Mon-Wed-Fri 1:30-2:20</p>
<h2>Honor code</h2>
<p>Do the right thing</p>
```

The CSS panel contains the following code:

```
h1, h2 {
  font-family: Arial;
}
```

On the right side of the browser, the rendered output is shown. It features a large heading 'Course Info', followed by a sub-heading 'Lectures', then the text 'Mon-Wed-Fri 1:30-2:20', another sub-heading 'Honor code', and finally the text 'Do the right thing'. The text is rendered in a serif font, consistent with the 'Arial' font-family specified in the CSS.

# Selector summary

Example	Description
<b>p</b>	All <b>&lt;p&gt;</b> elements
<b>.abc</b>	All elements with the <b>abc class</b> , i.e. <b>class="abc"</b>
<b>#abc</b>	Element with the <b>abc id</b> , i.e. <b>id="abc"</b>
<b>p.abc</b>	<b>&lt;p&gt;</b> elements with <b>abc class</b>
<b>p#abc</b>	<b>&lt;p&gt;</b> element with <b>abc id</b> ( <b>p</b> is redundant)
<b>div strong</b>	<b>&lt;strong&gt;</b> elements that are descendants of a <b>&lt;div&gt;</b>
<b>h2, div</b>	<b>&lt;h2&gt;</b> elements and <b>&lt;div&gt;</b> s

# Grouping selectors

## 2 Common bugs:

p.abc    **vs**    p .abc

p .abc    **vs**    p, .abc

- A <p> element with the **abc** class **vs**  
An element with the **abc** class that descends from <p>
- An element with the **abc** class that descends from <p> **vs**  
All <p> elements *and* all elements with the **abc** class

# Combining selectors

You can combine selectors:

```
#main li.important strong {  
  color: red;  
}
```

**Q: What does this select?**

# Grouping selectors

**Q: What does this select?**

```
#main li.important strong {  
  color: red;  
}
```

**A: Read from right to left:**

- `<strong>` tags that are children of `<li>` tags that have an "important" class that are children of the element with the "main" id.

# Colliding styles

When styles collide, the most specific rule wins ([specificity](#))

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

```
<div>  
  <strong>What color am I?</strong>  
</div>
```



# Colliding styles

When styles collide, the most specific rule wins ([specificity](#))

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

```
<div>  
  <strong>What color am I?</strong>  
</div>
```

# Colliding styles

Specificity precedence rules ([details](#)):

- `ids` are more specific than `classes`
- `classes` are more specific than element names
- Style rules that directly target elements are more specific than style rules that are inherited

# Colliding styles

- If elements have the same specificity, the later rule wins.

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

```
<div>  
  <strong>What color am I?</strong>  
</div>
```

Aside: The process of figuring out what rule applies to a given element is called the [cascade](#). This is where the "C" in *Cascading* Style Sheets comes from.

# Inheritance

We saw earlier that CSS styles are inherited from parent to child.

Instead of selecting all elements individually:

```
a, h1, p, strong {  
    font-family: Helvetica;  
}
```

You can style the parent and the children will inherit the styles.

```
body {  
    font-family: Helvetica;  
}
```

You can override this style via specificity:

```
h1, h2 {  
    font-family: Consolas;  
}
```

# Inheritance

While many CSS styles are inherited from parent to child, **not all CSS properties are inherited.**

```
a {  
  display: block;  
  font-family: Arial;  
}
```

`<em>` inherits the  
font-family property,  
but not display:

```
<a href="/home">  
  Back to <em>Home</em>  
</a>
```

[Back to Home](#)

# Inheritance

While many CSS styles are inherited from parent to child, **not all CSS properties are inherited.**

- There's no rule for what properties are inherited or not; the inheritance behavior defined in the CSS spec.

- You can look it up via MDN, e.g.

[font-family](#): Inherited yes

[display](#): Inherited no

- Generally text-related properties are inherited and layout-related properties are not.
- (You can also change this via the [inherit](#) CSS property, which is somewhat esoteric and not often use)

# <a> colors?

Hmm, MDN says [color is inherited](#)... but if I set the body color to deeppink, links don't change color:

```
CSS
body {
  color: deeppink;
  font-family: Helvetica;
}
```

```
HTML
<h1>Chocolate</h1>
<p>
  <a href="https://www.ghirardelli.com/">Ghiradelli</a>
  is not overrated
</p>
```

<a> inherits font-family...  
Why doesn't <a> inherit color?  
([Codepen](#))



# User agent styles

This is because the browser has its own default styles:

- Browser loads its own default stylesheet on every webpage
- Not governed by spec, but there are [recommendations](#)

```
<!DOCTYPE html>
<html>
  <head>
    <title>CS 193X</title>
    <!--
      NOT TOTALLY ACCURATE: This isn't actually injected
      in the HTML, but it is loaded silently!
    -->
    <link rel="stylesheet" href="user-agent-style.css" />
  </head>
```



# <a> colors?

So to style <a> links, we have to override the browser default link style by explicitly setting a color:

```
⚙ CSS
body {
  color: deeppink;
  font-family: Helvetica;
}

a {
  color: deeppink;
}
```

```
⚙ HTML
<h1>Chocolate</h1>
<p>
  <a href="https://www.ghirardelli.com/">Ghiradelli</a>
  is not overrated
</p>
```



# Link-related CSS

Since we're on the topic of links:

- How do we style **visited** links differently from **unvisited**?

# CSS pseudo-classes

[pseudo-classes](#): special keywords you can append to selectors, specifying a *state* or *property* of the selector

Syntax	Explanation
<b>a</b>	All anchor tags (links) in all states
<b>a:visited</b>	A visited link
<b>a:link</b>	An unvisited link
<b>a:hover</b>	The style when you hover over a link
<b>a:active</b>	The style when you have "activated" a link (downclick)

There are more [pseudo-classes](#) than this; have a look!

Before we move on:  
A few style notes

# Why not `<div>` everywhere?

Technically, you can define your entire web page using `<div>` and the `class` attribute.

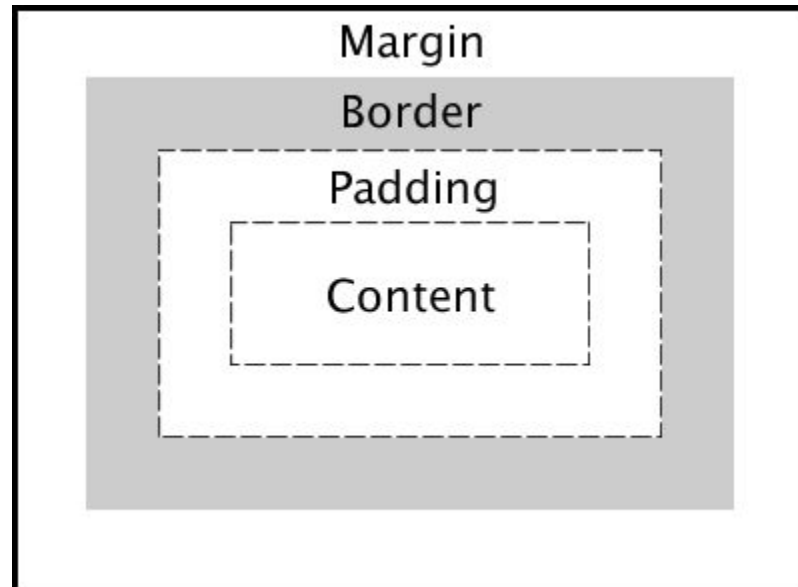
- Is this a good idea?
- Why does HTML have `ids` when you have `classes`?
- Why does HTML have `<p>`, `<h1>`, `<strong>`, etc. when you have `<div>`, `<span>`, `class`, and `id`?

# CSS Box Model

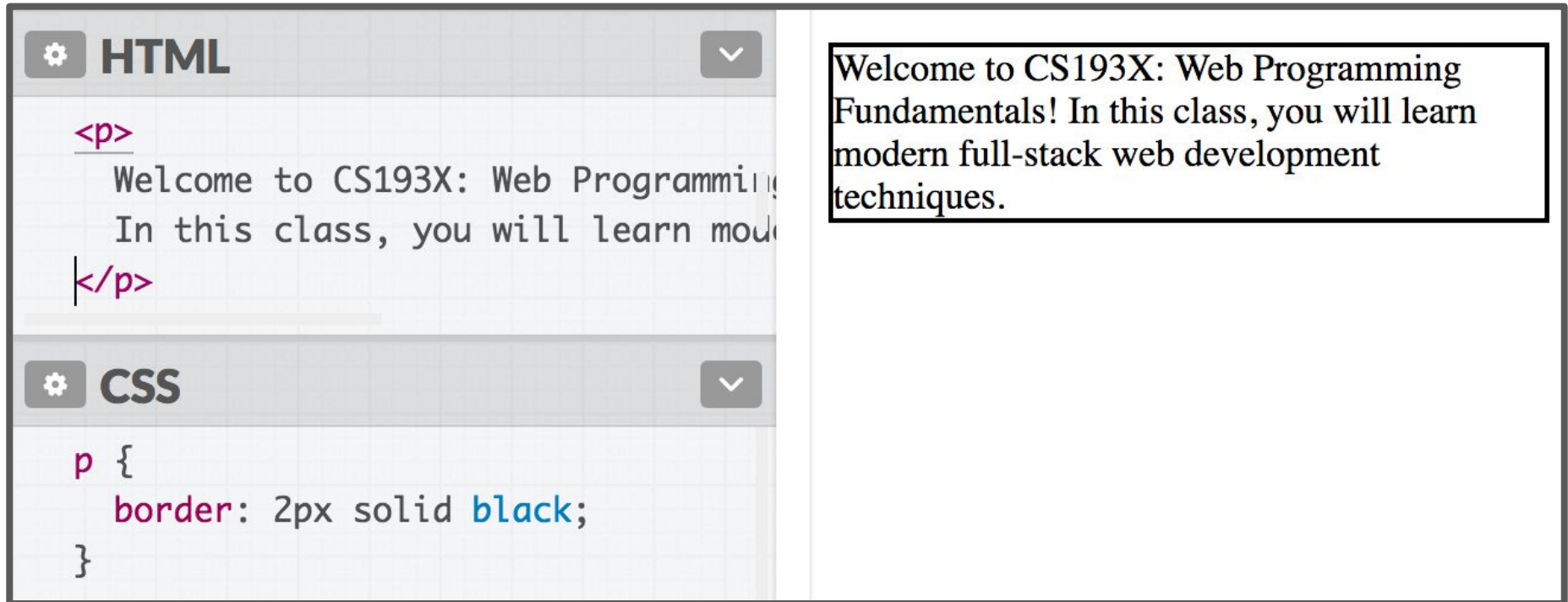
# The CSS Box Model

Every element is composed of 4 layers:

- the element's content
- the **border** around the element's content
- **padding** space between the content and border (inside)
- a **margin** clears the area around border (outside)



# border



We've used the [shorthand](#):

`border: width style color;`



# border

Can also specify each border individually:

`border-top`

`border-bottom`

`border-left`

`border-right`

And can set each property individually:

`border-style: dotted;      (all styles)`

`border-width: 3px;`

`border-color: purple;`

# border

Can also specify each border individually:

```
border-top  
border-bottom  
border-left  
border-right
```

And can set each property individually:

```
border-style: dotted;      (all styles)  
border-width: 3px;  
border-color: purple;
```

There are other units besides pixels (px) but we will address them in the next couple lectures.

# Rounded border

Can specify the `border-radius` to make rounded corners:

```
border-radius: 10px;
```

You don't actually need to set a border to use `border-radius`.

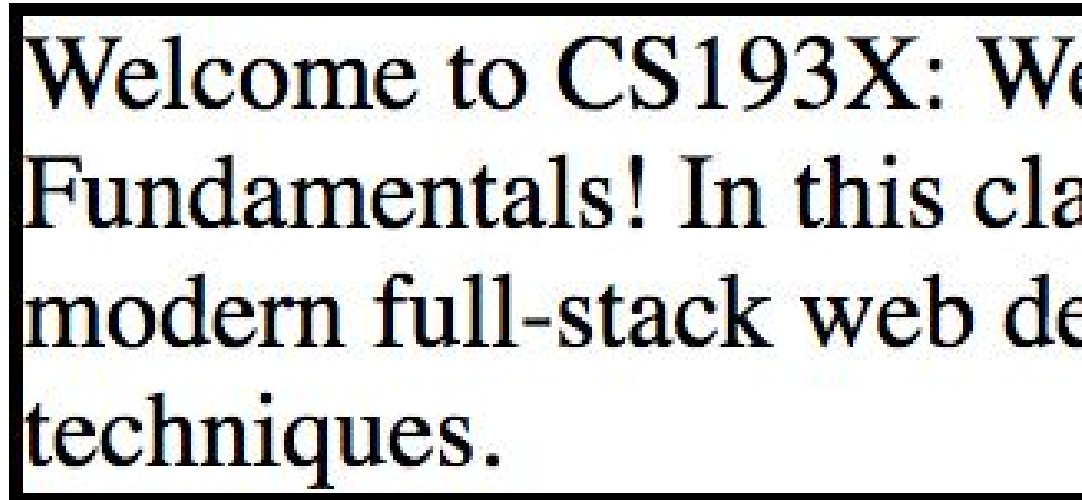
```
p {  
  background-color: purple;  
  border-radius: 10px;  
  color: white;  
}
```

Welcome to CS193X: Web Programming Fundamentals! In this class, you will learn modern full-stack web development techniques.

# Borders look a little squished

When we add a border to an element, it sits flush against the text:

**Q: How do we add space between the border and the content of the element?**



**Welcome to CS193X: Web Fundamentals! In this class, we'll learn modern full-stack web development techniques.**

# padding

```
p {  
  border: 2px solid black;  
  padding: 10px;  
}
```

Welcome to CS193X: Web Programming Fundamentals! In this class, you will learn modern full-stack web development techniques.

padding is the space between the border and the content.

- Can specify padding-top, padding-bottom, padding-left, padding-right
- There's also a [shorthand](#):

padding: 2px 4px 3px 1px; <- top | right | bottom | left

padding: 10px 2px; <- top+bottom | left+right

# <div>s look a little squished

When we add a border to multiple divs, they sit flush against each other:



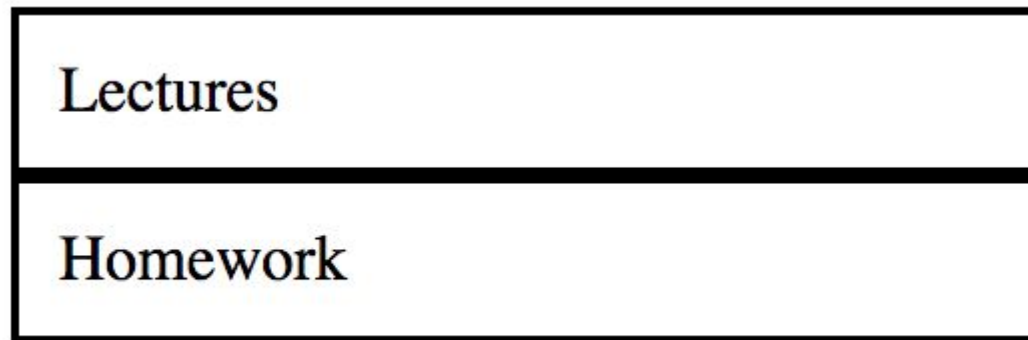
The screenshot shows a code editor with two panels. The left panel, titled 'HTML', contains the following code:

```
<div>
  Lectures
</div>
<div>
  Homework
</div>
```

The right panel, titled 'CSS', contains the following code:

```
div {
  border: 2px solid black;
  padding: 10px;
}
```

**Q: How do we add space between multiple elements?**



# margin

```
div {  
  margin: 20px;  
  padding: 10px;  
  border: 2px solid black;  
}
```

Lectures

Homework

margin is the space between the border and other elements.

- Can specify margin-top, margin-bottom, margin-left, margin-right
- There's also a [shorthand](#):

margin: **2px 4px 3px 1px**; <- **top** | **right** | **bottom** | **left**

margin: **10px 2px**; <- **top+bottom** | **left+right**

More box model:  
Next time!