

Lecture 11: Intermediate CSS and Accessibility

CS-546 – WEB PROGRAMMING



Recap and Intro

Recap

AJAX allows us to make network calls without leaving the page

The web is dark and full of security holes, but we can mitigate some of them.

This week

Keyboard navigation

Advanced CSS

<https://github.com/Stevens-CS546/Lecture-11.git>

Accessibility through Keyboard Navigation

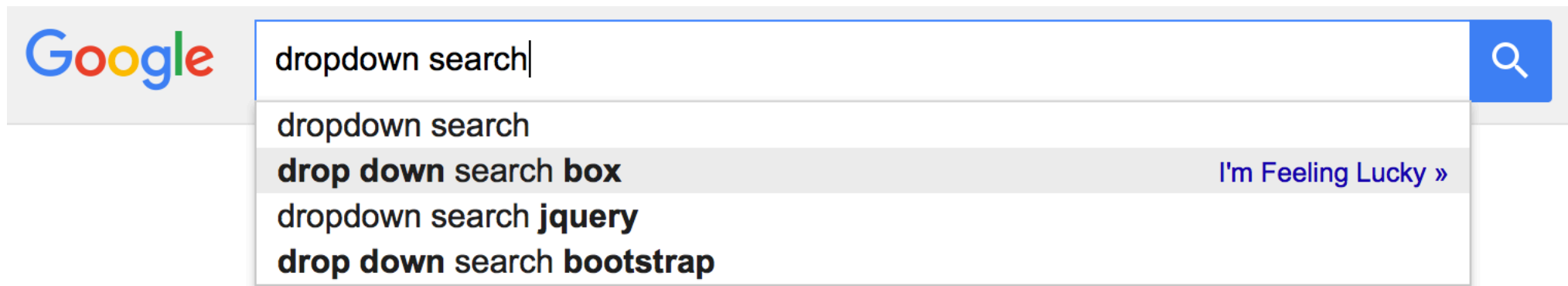
Complex frontends are hard to use

Many websites have search boxes that show results as you type in them.

Features like this are great; they save time, they are easy to use, and they simplify the process of using the web.

From an accessibility standpoint, to make this easy to use for *all* users, you need to do some work.

Features like this should be entirely usable via keyboard for accessibility reasons.



Case Study: live searches

The previous slide shows a 'live search', an adaptive search that shows results as they are found that you can select.

You can leverage the fact that your user is focused on an input to make the dropdown entirely keyboard navigable with ease!

- Add an event listener on that input for the *keydown* event
- Check which key was hit
 - if it was some sort of direction (up, down, left, right) then perform some logic to add a some indicator to a particular search result that is the selected
 - If it was `enter` then navigate to the selected search result, or a search page if none are selected
 - If it was `escape` then unselect any search results
 - If it was any other character, then simply re-query your search results and reset which result is the selected result

Using tabindex

The *tabindex* attribute allows you to focus on elements that can't normally be focused on, or change the order that you focus on elements

- This allows you to target an element using the `:focus` CSS selector

Your *tabindex* can be positive, negative, or 0:

- If it's positive, elements will be tabbed from the lowest to highest number of *tabindex*
- If it's 0, it will allow the element to be focusable but not change the order that you can tab into it; it would just be where it is in relation to the rest of the document (like static positioning)
- If negative, it will be ignored on tab but focusable
- Elements with an equal *tabindex* will be focused on in the order they appear in DOM

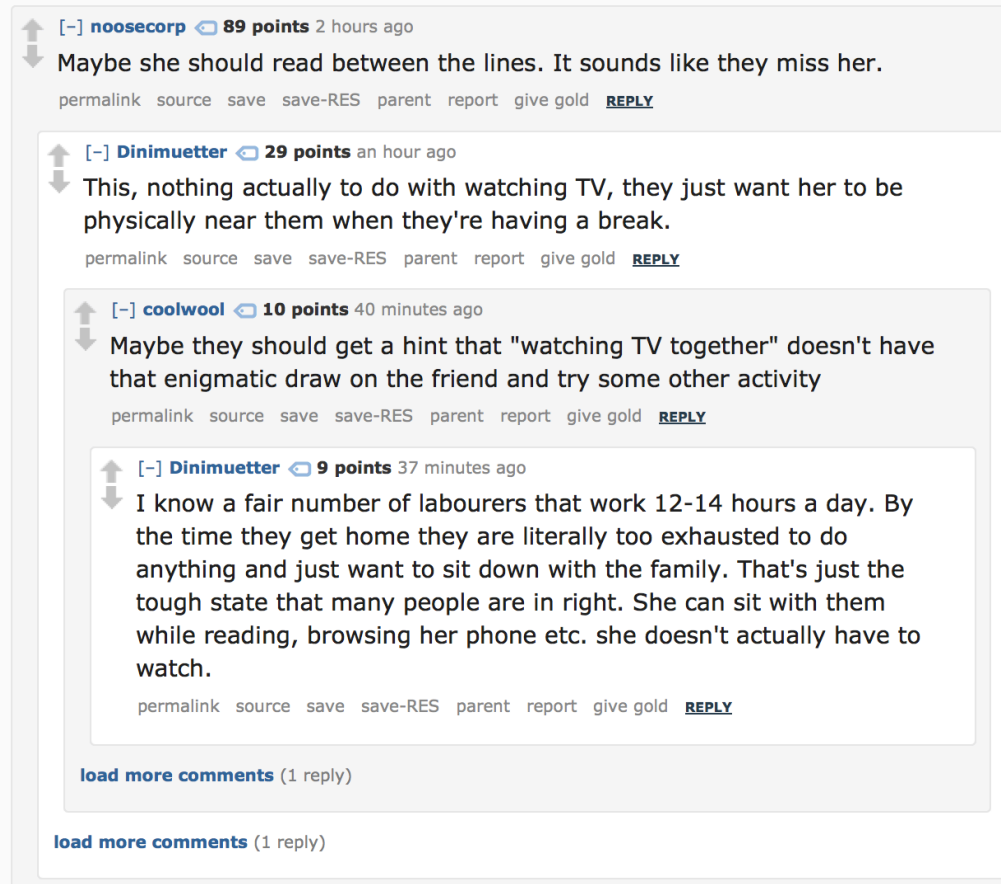
Case Study: reading comments on a website

Some websites are entirely devoted to reading comment chains, but they hide comments nested after a certain level.

While this will *probably* be entirely navigable through anchors, you would have to go through many anchors to get down to the first 'load more comments' factor.

Using a similar pattern as before, you can actually navigate through divs.

You would use the *tabindex* property, as well as some keyboard watching, to allow you to be focused on a particular post and navigate with the keyboard between posts.



The screenshot displays a comment chain on a website. At the top, a comment by user 'noosecorp' (89 points, 2 hours ago) is shown. Below it, a comment by 'Dinimuetter' (29 points, an hour ago) is visible. Under 'Dinimuetter's comment, there is a comment by 'coolwool' (10 points, 40 minutes ago). Finally, under 'coolwool's comment, there is a comment by 'Dinimuetter' (9 points, 37 minutes ago). Each comment includes a 'load more comments' link and a 'REPLY' link. The comments are nested, with each reply appearing below its parent comment.

[~] noosecorp 89 points 2 hours ago
Maybe she should read between the lines. It sounds like they miss her.
permalink source save save-RES parent report give gold [REPLY](#)

[~] Dinimuetter 29 points an hour ago
This, nothing actually to do with watching TV, they just want her to be physically near them when they're having a break.
permalink source save save-RES parent report give gold [REPLY](#)

[~] coolwool 10 points 40 minutes ago
Maybe they should get a hint that "watching TV together" doesn't have that enigmatic draw on the friend and try some other activity
permalink source save save-RES parent report give gold [REPLY](#)

[~] Dinimuetter 9 points 37 minutes ago
I know a fair number of labourers that work 12-14 hours a day. By the time they get home they are literally too exhausted to do anything and just want to sit down with the family. That's just the tough state that many people are in right. She can sit with them while reading, browsing her phone etc. she doesn't actually have to watch.
permalink source save save-RES parent report give gold [REPLY](#)

[load more comments](#) (1 reply)

[load more comments](#) (1 reply)

Practical: Making a Rich Text Editor

Here's a fun fact – you can make certain elements *editable* using the *contenteditable* attribute.

We can use this to make a contenteditable div that is entirely stylable via the keyboard, rather than buttons!

To start off, we will make a simple Rich Text Editor that will allow you to open and close emphasized, strong, and underlined text.

Advanced CSS

Media Queries!

One of the most powerful things about modern CSS is that we can use media queries.

Media queries allow us to conditionally apply styles.

We can easily apply based on:

- width / max-width / min-width
- orientation (portrait vs landscape)
- media type (screen, print, all, speech)
- more!

The basic format is, in your stylesheet:

```
@media (min-width: 700px) {  
    /* These rules will apply when the screen is 200px or wider */  
    .nav { width: 650px; margin: auto;}  
}
```

Print Layouts

You may use media queries to create styles that will apply to your page when it's printed only.

Often, you will use this to:

- Hide all but essential graphics
- Hide advertisements
- Create an easy to read version of your page that has little to no color
- Remove content that you would not want included, such as discussion comments

In your link tag, you can include a *media* attribute that describes which media type to apply those styles to.

- all
- print
- screen
- speech

Mobile First

You may be tempted at first to start changing your layout using media queries that change your layout as your screen size gets *smaller*.

Countless developers clocking in countless hours have come to the following conclusion: it's easier to define styles for your smallest screen, and add new styles that get applied as the screen gets larger.

An example, using a navigation that is a *nav > ul > li* setup:

- By default, your list items would display as blocks, each on their own line
- After a browser is 780px wide, they would float next to each other and display horizontally.

This is called *mobile first* development, in which you design for the smallest size first and support larger resolutions on top of that.

Transformations

In CSS, we can transform elements several ways:

- Translation
- Scaling
- Skewing
- Rotating

```
translated { transform: translate(10px, -40px) }
```

```
.rotated { transform: rotate(12deg) ; }
```

Animation

We can animate our elements rather easy, by defining and applying an animation:

```
.rotate this forever {  
  -webkit-animation: infinite_rotation 2s infinite linear;  
}  
  
@-webkit-keyframes infinite_rotation {  
  from {  
    -webkit-transform: rotate(0deg);  
  }  
  
  to {  
    -webkit-transform: rotate(359deg);  
  }  
}
```


Hardware Accelerated CSS!

Many animations can be accomplished by using JavaScript to manually manipulate elements. You can make extremely complex animations that way by treating it like a video game script, where you manipulate things frame by frame.

This, however, is extremely poorly performant because you cannot take advantage of your computer's native hardware acceleration.

When you define an animation in CSS using certain CSS3 rules, your browser can understand exactly what you are trying to do and use hardware acceleration. This makes CSS only transformations extremely smooth.

By using certain CSS3 factors during animations, such as the translation rule rather than using left / right values we can force hardware acceleration.

Hardware-Accelerated Properties

Only the following rules can result in hardware acceleration being used

- opacity
- translate (or translateX, translateY, translateZ, translate3d)
- transform: scale;
- transform: rotate

Pseudo-Elements

Pseudo-elements allow you to target the content of elements in more complex ways.

- First line
- First letter
- Before
- After
- Selection
- Placeholder

Content

You can add content with CSS, as well!

This is useful for:

- Adding things to certain text; ie, quotation marks to block quotes.
- Creating an icon set
- [Creating hovering tooltips](#)

Combining Content and Pseudo-Elements

It is very common to combine the CSS *content* rule with pseudo-elements in order to add and style things before and after elements.

Commonly, you'll see:

- Adding quotation marks before blockquotes using the `::before` pseudo-element that appear at the top left of an element, and one at the bottom right using the `::after` pseudo-element
- Creating icon sets
- Creating clearfixes

Transitions

When CSS properties change, we can set transitions that target those properties to make them occur over a period of time.

For example, you can set the font-size of an object to change over time when a class is added:

```
.big-text {  
    font-size: 60px;  
    transition: font-size 2s ease-in-out;  
}  
  
.big-text.smaller {  
    font-size: 30px;  
}
```

Questions?
