* Get lecture repo
* Slides
* PollEverywhere
* Bring up Visual Studio and load the lecture folder
* Bring up live-server and the Console

# Slide 2

Plan for today

I will talk for a bit about Stage 2... hopefully to make things clearer

# Slide 3

Project Stage 2

Object is to make your stage1 page interactive

Clicking, moving the mouse, etc... it's more than mouse-over. We want to see that when the user does something that the page changes. Not just with hyperlinks.

More than just mouse over transitions as well.

You are not locked into what you have done in stage 1. If you decide to move away from what you did in stage 1, that's fine, but you need to keep the same level of styling in place as what you did in your stage 1. It needs to be responsive, cool looking, have a grid, etc...

**Phase 2 is still an individual project.** This is so that everyone can show individually that they have acquired the skills from this course so far.

One thing to consider is that Phase 3 is when we start working with someone else. And naturally you will combine your work, so you might be better off waiting to change domains until that phase rather than now.

I am hoping to get grading done by Friday of this week. There are a lot of projects and it's time consuming. You will still have 1.5 weeks before phase 2 is due, and you should be able to make progress prior to getting the grade and feedback from phase 1.

1) **Fix the outstanding github issues from phase 1.** I'll give you some things to fix. Hopefully they are relatively minor. But resolving these issues is something you will need to do.

2) **Dynamically display external data**

You'll use AJAX requests to pull in this external data. This isn't too hard and is something like 2 lines of code. It's a bit weird syntax, but you'll get it.

The dynamic data can be real (like poll data for the election) or it can be mock up data from a spreadsheet or CSV file. It doesn't need to be from a public API. Feel free to generate some data on your own and save to a CSV if that is the most expedient and effective method.

You should start thinking about what data you want to pull in. You might need to change the focus of your app to find a way to use external data.

It’s perfectly fine if you talk with others in the class about data sets… particularly if you are going to be working with someone. You can all use the same data set.

3) **Enable user to interact with page and see content change**

Clicking causes the page context to change and not just having and on/off switch

You click on a trail from a list and a map shows up with where it’s located

Then you click on the button by the side of map and it shows a picture of the trail as well

**sequential** - you need to be able to click on something it changes the context, then click on something in the new context and it will once again change. You need to be able to **keep state** in a variable to do this. There will be a basic example of this in the lecture demo today.

The Shiny Data Tables example meets the requirement.

Polished and Frictionless - get it working smoothly.

If it takes time to do things, then you should use spinners to show that the page is loading.

You are welcome to create a new html page, but it's not required.

**~~4) Utilize another JavaScript Library~~**

~~This should be another 3rd party library. There are a ton of options here~~

~~Calendar, maps, plots~~

~~Figure out how to use it. Then integrate into your page.~~

5) Accessible, responsive, well written

Buttons should be big enough on a small screen.

Use the aria-label stuff to handle when you dynamically alter the DOM so that a screen reader can know

Look through the JavaScript Style requirements link

# SLIDE 4

# SLIDE 5

Use addevent listener rather than onclick. It’s better to keep concerns separate. HTML doesn’t need to know anything about javascript that is used. Instead keep it all in the javascript

# SLIDE 6

# SLIDE 7

Bring up the demo file in Visual Studio

The DOM is an API with the browsers interact.

In the developer console, look at “elements”, this is basically the DOM

We’re able to interact with these elements using jscript. We don’t actually change the HTML, but we are interacting with the DOM in memory as it is rendered on the screen.

"document" is the object. we call methods off document object.

.querySelector(cssSelector) is easier than .getelement both of these are available off of the document object.

querySelector() selects the **first** element that meets the selector.

querySelectorAll() that picks **all** the elements.

Usually we just pick one thing at a time though

You use put in the same CSS Selector syntax with.querySelector as we've already been using for CSS selectors

# SLIDE 8

Basic example of how this works:

First you need a reference to the element

You can get the text of an element with the .textContent() method

You can assign a new value to the element, simply by assigning elem.textContent="this is different content!";

You use the .innerHTML method to add styles to the text.

Be careful when using .innerHTLM. If the user is providing the content here, it's a place where script could be injected.

Note that the HTML file is not changed.

What is happening is that the script is loaded after the HTML has been rendered and the script alters the DOM in memory and the browser shows the modified DOM

# SLIDE 9

**Demo Example 1**

First, we use the 'h1' as our querySelector parameter to get a reference theH1 to the element

we can also grab a paragraph within the header. See how the query selector uses deceendent selectors here (‘header p’)

And update the HTML (formatting) by calling subtitle.innerHTLM = subtitle.textContent + "<em>Prof Carlson</em>"

textContent() – just changes the text

innerHTML() – changes the html as well

These are the ones to use for the best browser coverage.

The html file has not changed. The DOM got updated in memory.

# SLIDE 10

Demo Example 2

Agree that this selector grabs the image of the puppy img?

Change an attribute

We use the ‘.’ Property to grab a property on the elements.

Not every property allows you to change a property in this way. You can’t take any arbitrary attribute and change it this way. But the standard elements use this way.

# SLIDE 11

Example 3

Change a Style

We use the classList.add to add additional styles. You can remove too.

In the console type: img.classList.toggle('scale-out')

Twice

We’ll show how to do this with a button rather than with manually having it happen in the console

The image is still in the DOM, it’s just hidden.

# SLIDE 12

Example 4

Creating a new element

First, we create, then we set the innerHTML by assigning

It doesn’t show up yet however…

Then we add to the appropriate place in the DOM

Here we find the UL and append the newLi to the end of the classList

Again, be careful using .innerHTML()

You wouldn’t want to do this where the element could take <script> elements from user input.

# SLIDE 13

Use Java function to add items

Then call the function;

# SLIDE 14

Now you can iterate and quickly add items without typing a bunch of HTML

So we add the jar.innerHTML= ‘ ‘ (empty string) to clear the cookies before we start so that we don’t just add

So now we show 5 when we call it a second time

# SLIDE 15

Example 6

Here we want to show how to interact with the first button.

**addEventListener**(param1, param2)

First parameter is the **kind of event** (click, keypress, mouse up, mouse down, change events, blah blah blah)

Second parameter is the **callback function**

Here we're watching the button. When it gets clicked you can see it gets logged in the console

# SLIDE 16

Example 7

Keeping state as we toggle back and forth

Need a variable (isPuppy) to keep track of what's currently displaying

Example of state. So you can have a variable that you use to toggle.

Use the addeventlistener() method rather than the html.onclick

You can add/remove.

Html does support onclick but it’s a bit of a hack. Keep the java script in the javascript.

Don’t be inlining jscript in html. And even though you can do “onclick” in jscript. The other is better. More browser support.