1. **Intro to HTML**
    a. Many of you may have heard of or used HTML before, but it's not a pre-req and we know it may be new to some of you, so we'll start from scratch
    b. Stands for Hyper Text Markup Language: not quite a coding language, but pretty close. A formal structure to *mark up* the content of a web page to add formatting and make it easier to navigate
    c. HTML is made up of elements which can be nested, kind of like a row of boxes that can have other boxes inside. There are many different kinds of elements.
    d. Example of a single element: paragraph element. Walk through it (brackets, opening and closing tags)
    e. Example of nesting: ordered list element. Walk through it
    f. Relate this to the DOM:
        i. The Document Object Model is an actual code object that's instantiated as a tree. You can take the idea of boxes inside of boxes, and instead think of it as a tree with child and parent nodes, where each node is an HTML element. We evaluate the HTML code to produce the DOM object, just like you could evaluate a = 2 + 5 to get an object holding the value 7.
    g. What are the benefits of making pages in this way?
        i. Closely ties the structure of the page to its semantics (what it actually represents)
        ii. Easy to change element types to change functionality, easy for an application scanning the page to know what each element's functionality is
            1. Picking the right element types for the content you're representing
            2. Applications e.g. screenreaders for accessibility
        iii. Easy to apply code (e.g. formatting) to all of the elements of a type, which you probably want to have the same functionality
        iv. Nested style reflects how we actually think about most webpages - broken down into sections, which may have subsections, etc.
    h. Attributes:
        i. Classes: a class is basically defining your own sub-type of an element, so for example you might have paragraphs that are in the class 'quote', and paragraphs that are in the class 'response', and that lets you apply different formatting to e.g. a quote from your classmate's peer review and your response to it
        ii. IDs: an ID should be unique for the element throughout the entire file, because it lets you refer to that element specifically
        iii. Show the formatting of an attribute
    i. Basic HTML elements you'll want:
        i. Header: <h1> through <h4>
        ii. Paragraph: <p>
        iii. Any container:
        iv. Inline container:

v. Link: <a href=””>
    1. Href attribute is where the actual link goes
vi. Image: <img src=””>
    1. Src attribute is where you can put a link to an image either publicly available online, or a path to a file stored in your site’s directory
vii. There are tons more elements, you can find them through google if you’re looking for something specific or we’ll give some reference links at the end of the slides. In general, best to be as specific as possible

j. Structure of an HTML document
    i. Don’t have to worry too much about these details, just know that you have to include them to make your HTML work right, so if you’re starting an HTML page from scratch, need to add them! Can copy them off the internet
    ii. Doctype declaration
    iii. HTML element - the “biggest box”
    iv. Head element - stuff that doesn’t appear on the page
        1. This does a variety of things - some of it shows up outside the page, like the tab’s title. Some of it is used to help make the page appear properly, like the character set. Some of it is used by other code, e.g. search engines use the page’s language to help show you the right results.
    v. Body - the actual webpage

k. Exercise! (see slides)

2. CSS Intro

a. CSS: cascading style sheets. How you make all those HTML elements look pretty!
b. Made up of rules that affect types or groups of element
c. Show example: a set of rules applying to paragraph elements that make their text red. Walk through it (brackets, property colon value semicolon)
d. Three main types of element you apply rules to:
    i. By selector/tag type (e.g. p)
    ii. By id (#id)
    iii. By class (.class)
e. Tons and tons of different things you can do with CSS, different properties, etc. We’ll go over a few specific helpful things next. To get you started, some basics. Note that some of these can have different types of value for their properties - no easy way to know, reference sheets will tell you
    i. Color (takes a string or a hex color)
    ii. Height, width, font-size
        1. There are tons of confusing units of measure, stick to just a few: pixels is the simplest, means what you think. Em means font size, so you can define everything else relative to the font size. Sometimes you can do percents, which means the percent of the

parent element (e.g. defining a child as 50% of the width of the parent.

iii. Border (takes multiple things!)

f. How to include CSS in HTML
   i. There are three different ways.
   ii. You can include it inline, by setting an HTML element's style attribute to a CSS rule or set of rules. This is very much not recommended - it's not modular or easy to change at all, since you have to repeat it for every single element. It's also easy to miss that it's there or accidentally mistype it, since it's all inside a string.
   iii. You can include it internally, inside the same file but next to the HTML instead of integrated with it. To do this, you make a separate "style" tag that goes after the "html" tag, and put all of your rules inside it. This is less modular so it's not the best way, but if you need to check something for debugging, it's a nice quick option.
   iv. The third option is the best one and the one that actually gets used, which is to make a separate file with all of your CSS and include a link to that file in the head of the HTML. The link rel="stylesheet" will tell the HTML file to use that CSS file for its styling. This is very modular! Separation of concerns yay! Lets you use one CSS file for multiple HTML files, or import multiple CSS files into one HTML file.

3. CSS Positioning

a. So far we've mostly looked at how to change the appearance of elements using CSS. Now, we're also going to talk about how to change their positioning. This is a really complicated topic that could probably be its own recitation if we wanted it to, so we're going to try to just touch on the most important parts, and give you some resources for learning more.
b. Box model
   i. First thing you need to know is that in CSS, we treat every element as though it is within a series of boxes.
   ii. The content and the padding box are both within the element; then there's the border (which is what gets styled when you set the border property), and then there's another margin box
   iii. You can change each of these independently to change how much space the content takes up, how much space there is between the content and the border (padding), and how much space there is between different elements (margin).
c. Positioning basics
   i. We won't actually go much into detail about these for two reasons. One, they're more self explanatory, and like other CSS properties, you can look up reference guides to them. Two, they often don't provide the solution you need. Instead, we'll cover two more complicated systems for layouts that may be more useful for complex designs.

ii. The main basic you need is ‘display’: this controls which layout system you’re using. You can set it to some basic things, like ‘inline’ which displays in the same horizontal space as other elements, or ‘block’ which takes up the full horizontal space. Or you can set it to one of the two systems we’ll be looking at in more depth.

d. Grid
    i. Both of these systems apply to individual elements, which means you can use both of them, or each of them more than once, within the same page. The idea is that you apply the system to a parent element, and then can use that system to choose the layout of all of the child elements
    ii. The grid layout is exactly what it sounds like: it applies a grid on top of your parent element. You can then specify which cells on the grid your child element should occupy.
    iii. Show the syntax for the parent element: setting display to grid and the size of the rows and columns
        1. Talk about “fr” unit meaning fraction
    iv. Show the syntax for the child elements: grid-column, grid-row
        1. Talk about how we count grid lines, not grid cells
e. Flexbox
    i. Grid is great for when you want to precisely position a set of things that may be of very different sizes, not related to each other in size/position, etc.
    ii. Sometimes that’s a lot of work to specify, and you have a bunch of things that are very similar and you want them all to appear next to each other.
    iii. In that case: flexbox!
        1. Lots of common use cases: e.g. if you have a blog theme that lists all your assignments as little pictures that are next to each other, that’s almost certainly a flexbox.
        2. Flexboxes are also great for centering things, which is famously very difficult with CSS
    iv. Show the syntax for the parent element: setting display to flex. That’s it!!
    v. If you want a very basic flexbox, that’s literally it. But there are a few more options. Most importantly, setting the axis!
        1. Flex-direction sets axis: there are four options. Row and column, and row and column reverse - these show the first item to the far end of the axis (e.g. you might want this if your items are listed chronologically but you want them to appear reverse chronologically)
        2. Once you’ve set the axis, you can align items along that main axis, as well as along the other secondary axis
            a. There are a lot of options here, not just front and start - also multiple ways to center the options, which is something you might want!

4. **Exercise!** (see slides)