This is the first main tutorial in a series on web development fundamentals, which will discuss html and the document object model, as well as css and svg. The goal of this tutorial is **not** to give you a comprehensive overview of these topics, but to give you enough of a starting point that you will be able to work on your projects and understand the (many!) external resources available on web development topics.

This tutorial will get you to the point of placing a line of static elements onto a page. They won't update dynamically. They won't respond to your interactions. They won't be organized into a nice layout. This is still to come. This tutorial is all about adding static elements to a page, and controlling their 'styling'.

1. HTML

HTML stands for Hypertext Markup Language. HTML defines elements on a web pages, in a hierarchical structure. In the early days of the internet, HTML allowed web developers to create largely static pages that could be linked to other pages (hypertext), and marked up to structure the page. The elements in html thus allowed developers to create pages with titles, headers, text, images and links to other pages. Today, we can use it to incorporate much more- graphics, scripts to dynamically update the page, buttons and UI elements, audio, video, etc.

In this class we aren't creating web pages, but **web applications**- programs that can respond to complex inputs from users, present graphics, and dynamically update. Not that long ago, it was very difficult to create complex applications on the web, but with modern web standards and contemporary browsers we can create interactive, dynamic applications and share them easily.

Elements, tags and attributes

An HTML page consists in a set of elements. Elements are indicated by tags, which begin and end the element.

```
<p>This is an HTML element.</p>
```

The 'p' tag is used to identify this element as a paragraph. Note how there is an opening tag (```<p>```) and a closing tag (```<\p>```).

Elements can be nested within other elements.

```
<p>This is an HTML element.  I have some text I want to <strong>bold and <u> underline <\u> <\strong> and other text that is not bolded or underlined</p>
```

Within a tag you can include 'attributes' about the element. For example, suppose you want to include a link within your text. You could write:

```
<a href='https://www.uc.edu/'>University of Cincinnati</a>
```

The attributes we will use most often are classes, ids, and style. We will discuss these in the next section.

HTML Boilerplate

Every HTML project you create will need some boilerplate- you can use this every time you get started with a blank html page.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8"/>
    <title>Ceci n'est pas un document HTML</title>
  </head>
  <body>
  </body>
</html>
```

The first line is a document type declaration. Then you have your root element - your html element. HTML5 documents need a header, which specifies a charset (utf-8).

Just as a note: some elements do not have internal content, and to avoid making developers type a closing tag every time they use these elements, they can instead replace

```
<foo></foo>
```

with

```
<foo/>
```

DOM

DOM stands for **Document Object Model**. When the browser parses the html document, it creates a hierarchical (tree-structured) model containing and connecting your page elements. This tree has a root ("html' element). Elements will have a 'parent', 'parents' can have multiple children. Children can have their own children, and will be a parent to those children.

This model creates a structure for accessing and modifying elements - changing the content, changing the style, adding and removing elements- using javascript or d3. For example, suppose you have created a section of your page within a 'div', and you want to access all child elements within that div. Javascript provides ways to select an element (by class or id), and then you can select all child elements of your selection.

You can (and should) inspect the DOM in your live browser when you develop. You can see the elements, their children, which is very helpful when debugging your code.

Let's give it a try. The following instructions work in the Chrome browser, which will be the default browser for this class.

1. Go to this page: http://cscheid.net/courses/spr15/cs444/lectures/week2/index.html#

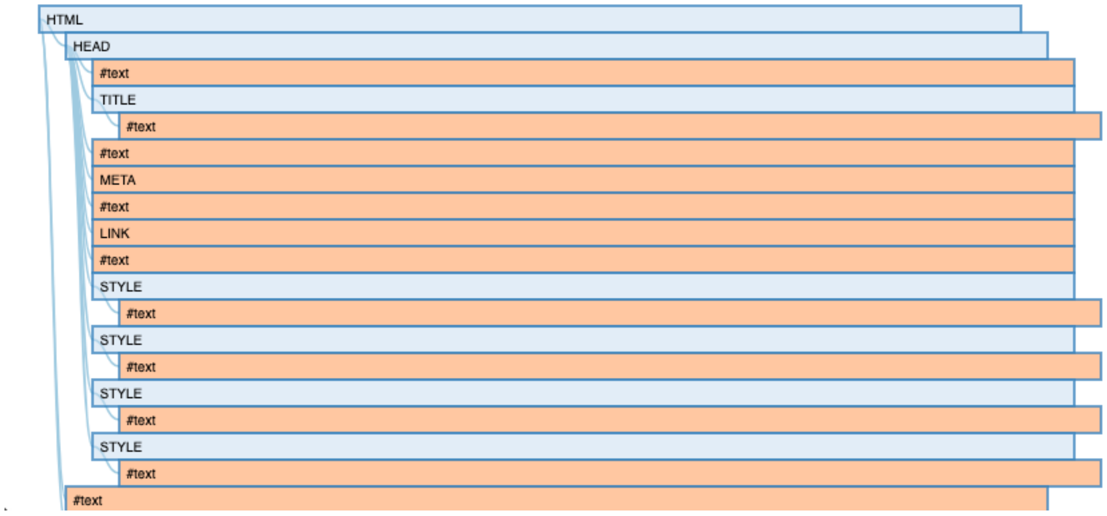
2. Right click on the page and select 'Inspect'

3. A small panel will open on the right of the page (or the bottom, depending on your settings).

***foo*, bar, baz**

DOM Visualization

Go!



4. Hover over the head element or the body. It will be highlighted on the page.

5. You can also select these elements (clicking on the small arrow to the left of the element), and see it's children.

6. Play around with this for a while, taking note of the structure of the page and the elements in the page. FYI- it contains a

7. You can also edit these elements live. Try editing the text in the title.

Elements

There are lots of elements you can add to your page.

You can add text elements in several different ways. You can use <h1> tags or <h2> tags, to create text elements with larger font sizes. You can use paragraph tags <p> to create other text.

You can add buttons of different kinds. Let's look at some examples NOTE- I inserted images of these elements, not actually clickable elements. I can't do this in canvas. So don't click the buttons! Even though they say 'click me'.

```
<button type="button">Click Me!</button>
```

How about a special button, like a play button? You can look up unicode symbols here:

```
<button type="button">&#9658;</button>
```

▶

Checkboxes, which need both a label and the input defined:

```
    <input type="checkbox" id="checkbox1" value="select">
    <label for="checkbox1"> Select me</label><br>
```

☐ Select me

Image elements:

```
    <img src="img/parisPhoto.jpeg">
```

There are LOTS of element types. I generally look these up at w3 schools, and copy and paste the starting point template into my code, and then edit to personalize the content.

Grouping html elements

If you add the elements in order, they will mostly appear in order and their layout with respect to other elements will either be:

Inline- which means they are next to each other on the page, in one row.

Block- which means that are stacked on top of each other like blocks.

Elements have default block or inline behavior. You can control some of this with elements: **div** or **span.**

Div's partition your elements into groups, and the elements within a div are treated as belonging to the same block, so they will be separated from other elements in a different line.

Span forces elements that may want to be in separate lines to be in the same line.

Let's look at an example:

Suppose you have this html code- which has a header text, and image and 2 buttons. The header text is a block element, it will occur on a separate line by default, but the buttons will be placed in line with the photo by default. As shown here:

```
    <h1> This is my amazing photo </h1>
    <img src="img/parisPhoto.jpeg">
    <button type="button">Add</button>
    <button type="button">Remove</button>
```

This is my amazing photo

Add Remove

How can I use a div to move the buttons onto a new line, beneath the photo?

```
<h1> This is my amazing photo </h1>
    <img src="img/parisPhoto.jpeg">
    <div>
        <button type="button">Add</button>
        <button type="button">Remove</button>
    </div>
```

This is my amazing photo

Add Remove

t

Classes and IDs

As you add elements to your page, you need ways to reference them through selectors, so you can modify or apply style rules, which apply to the appearance and behavior to selected elements. There are 2 ways to do this:

- classes, which reference a set of elements, and
- ids, which uniquely reference one element

Here is a div element with no class or id.

```
<div>User Interface Design Course</div>
```

If we want to reference this 'div', and modify it, we can select it through a unique id.

```
<div id="fall2022UI">User Interface Design Course</div>
```

If we have a set of elements in the page that we want to be able to select and modify as a group, we use a class.

```
<div class="courses">User Interface Design Course</div>
<div class="courses">Data Visualization Course</div>
```

We can use both ids and classes together for one element, as in the example below:

```
<div id="fall2022UI" class="courses">User Interface Design Course</div>
```

We can also use multiple class labels.

```
<div id="fall2022UI" class="courses ugrad grad">User Interface Design Course</div>
```

How will this be useful for our projects? Identifiers are particularly useful for labeling buttons, checkboxes, sliders or other controls that uniquely give a particular type of input from the user, and that you will want to capture. For example, suppose you have a set of buttons that update the interface when clicked. Perhaps you want two checkboxes, one labeled 'undergraduate' and another labeled 'graduate', and clicking on these checkboxes controls which content is displayed. You need some way to capture the inputs from these uniquely identified checkbox elements in your page, and labels from identifiers will help with this. If you want to apply style rules (size, color...) to these two checkboxes, you can use a class label to modify both checkboxes at once.

2. CSS

HTML creates the structure of your webpage- what elements are present, and how are they nested within other elements. CSS defines the style rules for your page- the colors, fonts, backgrounds, margins between elements, orientation of elements... These style rules can be set as attributes within individual elements. But this can get tedious and hard to manage. So, instead you can set style rules that apply across all elements of a certain type (Eg. how should all buttons in a page be styled?), or across all elements in a particular class or for specific elements with a particular id.

To include a style sheet in your page, you need the following line of code:

```
<link rel="stylesheet" href="css/style.css">
```

You can set styles inline in html. You can use a style element in the header, to define styles, but then these are not re-usable across different pages. It is better though to do this in your css file and link it in.

Style rules are typically formatted like this. You have a selector, followed by curly braces and a list of properties that apply to that selector.

```
/*this is a comment*/
selector {
    property: value;
    property: value;
}
```

Selectors can be elements (like div, h1, p, button....). They can apply to classes and to id's as well.

Let's see a style rule applied to all divs on the page:

```
div {
  border-radius: 25px;
  background: orange;
  padding: 20px;
  width: 200px;
  height: 150px;
}
```

Suppose you then had this html code in your page:

```
<p>Rounded corners for an element with a specified background color:</p>
<div>Rounded corners!</div>
```

Here's the result once the style sheet is parsed and the style rule is applied to your div.

cssExample1-1.png

The above example applied style rules to **all** divs in the page. If you want to apply style rules to **some** elements, but not others, you can use classes and identifiers.

Here's an example. Suppose you want to define a class of div that will have rounded corners. And you also want classes of div with a green background and others with a white background and blue border. You can define style rules for these different classes using the format: .class-name {...}.

```
.rounded {
  border-radius: 25px;
  padding: 20px;
  width: 200px;
  height: 150px
}

#rcorners1 {
  background: #73AD21;
}

#rcorners2 {
  border: 2px solid blue;
}
```

And the html:

```
<p>Rounded corners for an element with a specified background color:</p>
<p class="rounded" id="rcorners1">Background is green!</p>
<p>Rounded corners for an element with a border:</p>
<p class="rounded" id="rcorners2">Blue boarder!</p>
```

From the above, classes are selected in css with a dot, as in .className. In the above example, this means all the div's in class 'rounded', can be styled to have rounded corners. And then div's can be individually styled, through identifier selectors, using #idName.

The result:

Screen Shot 2021-12-28 at 5.04.12 PM.png

Here's another example. Suppose you have 3 types of buttons, with 3 different colors. You can define a base class for buttons, which will apply to all buttons. This can include text color, font, padding around the text.... etc. THEN you can create particular style rules for the different kinds of buttons, including different colors.

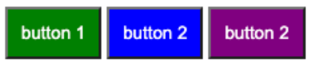
Let's first look at the style sheet:

```
.btn {
    padding: 10px;
    color: white;
}

.btn1 { background-color: green; }

.btn2 { background-color: blue; }

.btn3 { background-color: purple; }
```

Now let's look at the html:

```
    <button class="btn btn1"> button 1 </button>
    <button class="btn btn2"> button 2 </button>
    <button class="btn btn3"> button 2 </button>
```

What do we get?



Note- you can include style rules in your html page, in the header section. This would produce the same result for the buttons shown above:

```
head>
    <meta charset="UTF-8">
    <title>My Smart Fridge UI</title>

    <!-- The following line references an external css file -->
    <!-- <link href="css/style.css" rel="stylesheet"> -->
    <style>
        .btn{
                padding: 10px;
                color: white;
            }

        .btn1{ background-color: green;}

        .btn2{ background-color: blue; }

        .btn3{ background-color: purple; }

    </style>

</head>
```

Class selectors are used most often. But- What if you want a style rule to apply to just one element- exactly one button, or exactly one div or h1 element?

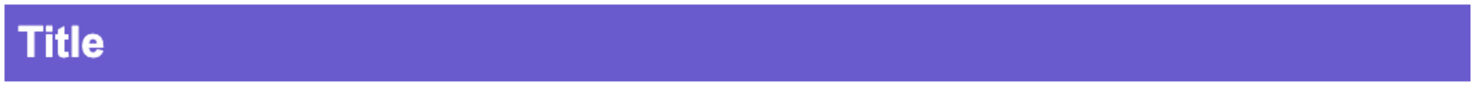
You can give that element an identifier, and then define the style rules using #i. Here's an example:

```
   <h1 id="main-header">
        Title
    </h1>
```

And the style rule:

```
#main-header{
    color: white;
    background-color: slateblue;
    padding: 10px;
    font-family: Arial, Helvetica, sans-serif;
}
```

Result:


Title

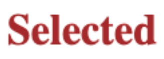
Now you may be asking- what if I want to combine selectors together in my css? There are lots of different rules for how selectors are combined together.

Suppose you want to set style rules for paragraph elements in a div, but not other paragraph elements on the page. There is a way to do that. Take this example:

```
div.main-div h1.brown{
    color: brown;
}
```

And the htmL:

```
    <div class="main-div">
        <h1 class="brown">
            Selected
        </h1>
    </div>
```


Selected

You can share style properties between selectors, by putting a comma between them. This can avoid duplication.

The take-away- css lets you style elements. You can set rules on elements, on classes and on ids. You can combine selectors in different ways to organize your style rules.

References:
* w3Schools is a good reference for css [at this link](https://www.w3schools.com/css/default.asp).

Want to do more?

This is a nice tutorial:

3. SVG

SVG stands for 'scalable vector graphics'. It is part of the HTML5 standards and it allows developers to add graphical elements to their page- shapes, lines, etc. D3, the visualization library we will use in this class, uses svg extensively to map data to graphical elements on the page. But, you can also use svg independent of d3.

Note: alternative methods for creating graphics on webpages include using the Canvas (Links to an external site.), which you can think of as a drawing area, or using WebGL (Links to an external site.), which includes the ability to draw elements in 3D. We are focusing on svg and d3 in this class, because they are the standard for contemporary work in creating 2D graphics, and d3 includes many powerful tools for presenting data visually.

SVG drawing basics:

- SVG is defined using markup code similar to HTML.
- SVG elements don't lose any quality when they are resized.
- SVG elements can be included directly within any HTML document or dynamically inserted into the DOM with JS.
- Before you can draw SVG elements, you have to add an `<svg>` element with a specific `width` and `height` to your HTML document, for example: `<svg width="500" height="500"></svg>` .
- The SVG coordinate system places the origin (0,0) in the top-left corner of the svg element. This means your coordinates go from **(0, 0) in the upper left corner**, to **(width, height) in the lower right corner. To put an item in the center, specify that is sits at (width/2, height/2). An item at (50, 100), will be 50 pixels to the right of the upper left corner, and 100 pixels down.**
- SVG has no layering concept or depth property. The order in which elements are coded determines their depth order.

Here is an example DOM with an svg element, width 400 and height 50. We can add basic shapes and text, setting the position and parameters of the elements.

```
<svg width="400" height="50">

<!-- Rectangle (x and y specify the coordinates of the upper-left corner -->
<rect x="0" y="0" width="50" height="50" fill="blue" />

<!-- Circle: cx and cy specify the coordinates of the center and r the radius -->
<circle cx="85" cy="25" r="25" fill="green" />

<!-- Ellipse: rx and ry specify separate radius values -->
<ellipse cx="145" cy="25" rx="15" ry="25" fill="purple" />

<!-- Line: x1,y1 and x2,y2 specify the coordinates of the ends of the line -->
<line x1="185" y1="5" x2="230" y2="40" stroke="gray" stroke-width="5" />

<!-- Text: x specifies the position of the left edge and y specifies the vertical position of the baseline -->
<text x="260" y="25" fill="red">SVG Text</text>

</svg>
```

And the result:

Screen Shot 2022-01-01 at 5.49.04 PM-1.png

Paths: Paths can be used to create more complex shapes, though it is not standard to manually specify these shapes very frequently. It essentially consists in a set of points, where a line is drawn from the first point to the next, and you can indicate when to close the shape. There also are ways to create paths using curves.

This path tutorial (Links to an external site.) may prove a useful reference.

SVG Grouping and Transformations:

Elements can be grouped together, and then acted upon together. This includes changes to presentation attributes, as well as geometric transformations (moved, rotated, etc). SVG has a transform attribute, the allows you to specify how the group of elements are collectively transformed.

Let's do an activity: HTML, CSS, SVG In-Class Activity 1 (Monday Sept 12)