

Module 3: HTML and structuring web pages.

Module 3 Objective: Learn the fundamentals of HTML and begin building the foundations of your digital portfolio.

What is HTML: HTML, or hypertext markup language, is a set of commands used to format documents that are served through the web browser.

NOTE: HTML is NOT a programming language—it is a markup language. Programming languages (Javascript, Python, C++, etc) use logic to automate processes, react to various conditions, and are chiefly concerned with taking data inputs and providing action or data outputs. Markup languages, on the other hand, are about organizing and displaying web documents on a browser.

Any interactivity you see on the web is a result of the integration of Javascript (JS), which is a programming language that works within the browser. HTML without JS basically looks and feels like the old school 90s web pages that simply display information without any action.

An HTML Page is organized thru the use of HTML elements (tags); some elements are used to designate the type of content they enclose:

```
<p>Hello</p>
```

In this case we have the text 'hello' designated as a paragraph, which is simply a way to say this is text that will print to a screen.

Other tags close themselves:

```

```

This image tag basically points to a specific image location through the attribute src (or source) and closes itself. This will result in the image located at that URL to show up on our web page.

In this case **src** is an attribute, information stored within tags that would affect what and how they display. Not all elements need attributes, but we will go into some commonly used tags and attributes within this module.

HTML PAGE STRUCTURE

All HTML documents contain the same basic structure

```
<html>
  <head>
    <!-- Here you put metadata like title and tags you want
    associated with the page for web browsers. You can also link your CSS
    styling here to add design to the page. What exists in the head section
    should NOT show on the page -->
  </head>
  <body>
    <!-- the body section on the other hand is where the content of
    the page will go and can be further broken down using divs (which we will
    explain shortly)-->
  </body>
</html>
```

Some quick notes and tips:

- Often, you will see pages having a <header> tag. Please note that the header is not the same thing as the head. Header sections are just a way to separate and organize the contents that show on the screen into various sections, and are basically the same thing as the <div> tag we will see later. You may also see pages with <footer> tags that serve the same function.
- The head tag is ONLY used for meta-information to inform browsers about what content is on the page and for linking styling sheets to change page design.
- Indentation is extremely important. HTML is basically tags nested within other tags, so how you indent those tags to organize them matters.

With proper indentation:

```
<body>
  <div>
    <p>Hello</p>
  </div>
  <div>
    <p> This is good</p>
  </div>
</body>
```

As opposed to:

```
<body>
<div>
<p>Hello</p>
</div>
<div>
<p> This is bad</p>
</div>
</body>
```

This may seem trivial, but when you have a large page with several sections, and sections within sections, and forms and images and headers and footers, it will be impossible to know what content is where without proper indentation.

A note on the head section and what typically is inside of it:

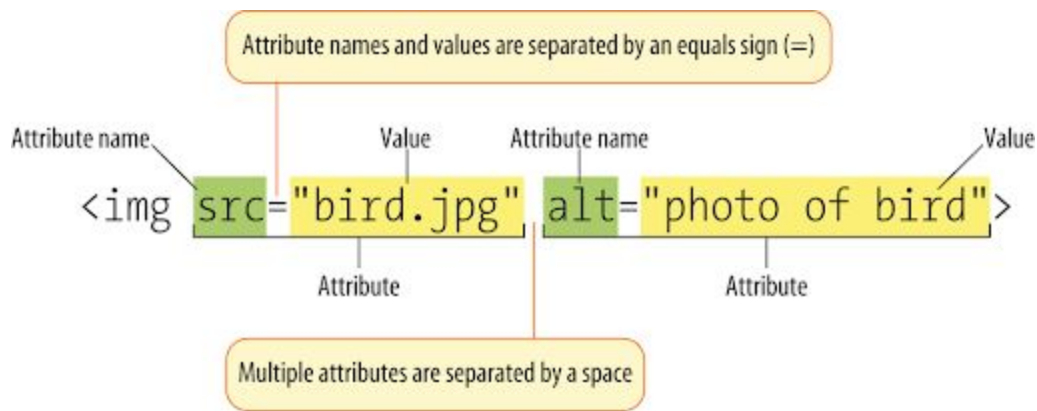
Here is a typical head section:

```
<head>
  <title>My Portfolio</title>
  <!-- this is the text that shows up in a browser tab --->
  <link rel="shortcut icon" type="image/png" href="icon_file_name.png"/>
<!-- this is the icon image that shows up in a browser tab --->
  <link rel="stylesheet" type="text/css" href="/style.css"/>
<!-- this is the link to a remote style sheet --->
</head>
```

As you can see in the above example, none of this stuff is content that actually shows up on the page. Instead, it's all info about how the browser should display what is in the body section.

Anatomy of a Tag

Before we continue, I think it is important we touch a bit on tags and attributes. As mentioned before, tags are all about letting the browser know what kind of content you are displaying and organizing that content. However we can alter the kind of content we display, or we can name specific content or sections to apply styling to via attributes.



Some common attributes are:

href : the URL that a hyperlink points to for use in the link tag `<a>`

src: the attribute that references the location for an image to display in an image tag

class: A way to name a grouping of elements, so that they can be selected to apply specific styling too.

id: Similar to class but only used to name one single HTML element to be referenced for either specific styling or to be tied to a JS action enabling interactivity.

An Example:

```
<head>
  <style>
    p {color: black}
    .div1 p {color:red;} //classes are identified with a .classname
    .div2 p {color: blue;}
    #yell p {color: yellow} // ids are identified with a #idname
  </style>
</head>
<body>
  <div class="div1">
    <p>Hello</p>
  </div>
  <div class="div2">
    <p>Goodbye</p>
    <p id="yell">AH!</a>
  </div>
  <p>Yo</p>
```

```
</body>
```

In the above example, there are several conflicting styling instructions. The first states that all text that is wrapped in a P tag should appear black; however, we next have two class-specific styles for div1 and div2 classes: CSS selects a class with a . last we have instructions for p tags with the id yell, where ids are selected by CSS with a #.

So what you would see here is:

Hello
Goodbye
AH!
Yo

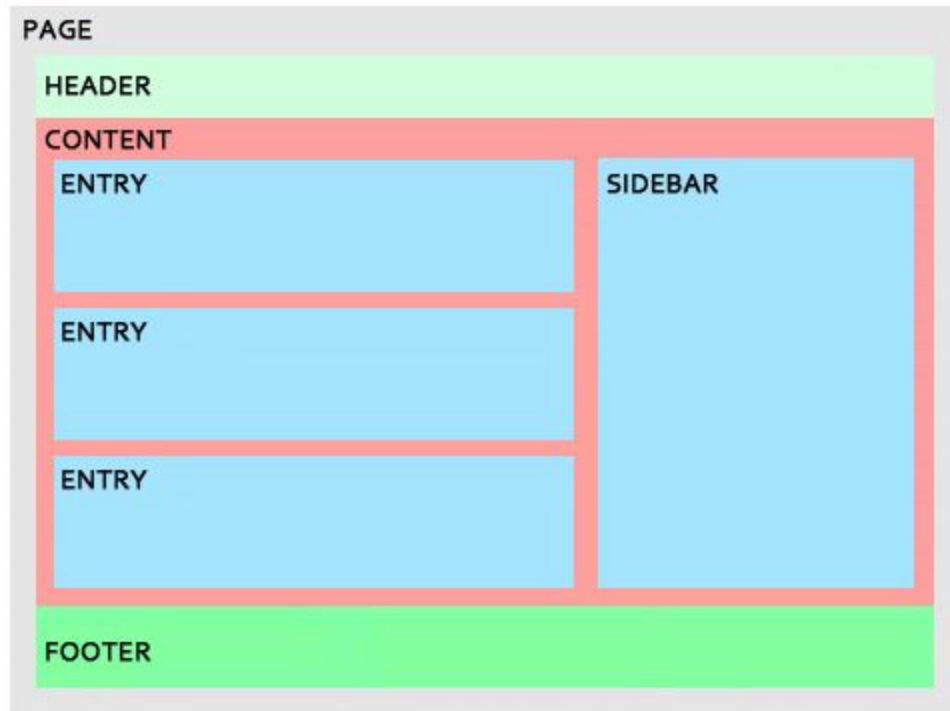
We will go into styling more next module. The purpose here is just to allow you to see the way classes and id's are used.

NOTE: classes can be reused; however, Id's should only be applied to one thing.

Organizing content into sections with Divs (dividers)

Before we go into some of the more commonly used HTML element tags we should talk about how certain tags can be used to section information off on a page.

Typically, we are all familiar with a web page and the fact it is not all one giant block of text, but is rather divided into sections either for specific content or styling.



In the above example we have a site layout, each of these subsections could be grouped in a div, and the sections then manipulated to be laid out a certain way based on CSS styling.

Note you will see html elements such as <content> <header> <footer> <section> etc. these all do basically the same thing as a div and are a layover from when styling was applied element to element.

Now through classes you can simply create a bunch of <div> tags and apply classes to them.

Example:

```
<div class="header">
  <p>My Blog Title</p>
</div>
<div class="post">
  <p> a post</p>
</div>
```

```
<div class="post">
  <p> another post</p>
</div>
<div class="footer">
  <p> Footer stuff </p>
</div>
```

You can also nest divs into each other:

Ex:

```
<div class="post-container">
  <div class="post-title">
    <p> Post title 1</p>
  </div>
  <div class="content">
    <p> Post Content blah blah blah</p>
  </div>
</div>
<div class="post-container">
  <div class="post-title">
    <p> Post title 2</p>
  </div>
  <div class="content">
    <p> Post Content blah blah blah</p>
  </div>
</div>
```

As you can see, we can add classes to divs so that we can include styling specifically for those divs. For example we may want to make the title text a larger font than the body text. In the case of the above example, that would look like:

```
.post-title p{
  Font-size: 24px;
}

.content p{
  Font-size: 14px;
}
```

We will be going more into styling and CSS later, but for now, just realize that classes are acting as specific labels for the divs so we can apply different styles to the same elements and sections based on their labels.

Common Tags You will Use

So far we have looked at a lot of the structural elements you will use as you build web pages and web application interfaces with HTML. However you also will be showing content, not only organizing it, so it will be important to become familiar with some of the common elements you will have to work with.

First off there are the common text elements:

```
<h1>Big Header</h1>
<h2>Not so big Header</h2>
<h3>Even less big Header</h3>
<h4>Small Header</h4>
<h5>More Small Header</h5>
<h6>Smallest Header</h6>

<p>Regular text</p>
```

So these text elements are a holdover to a time before CSS, where if you wanted different size headers you would either need to use one of the header variants (1 being the largest and 6 being the smallest) or you would need to specify a font-size attribute in the element itself.

However, now these things are less relevant. It still makes sense to use an h1 tag for titles and p for regular text, so that web crawlers can differentiate between title and content text, but the font sizes for all these elements can be altered using CSS so h1-6 differences are much less needed.

Media Elements:

```
 <!-- hosted on the same site -->
 <!-- hosted on an
external site -->
<video width="320" height="240" controls>
```



```
<source src="movie.mp4" type="video/mp4">
<source src="movie.ogg" type="video/ogg">
Your browser does not support the video tag.
</video>
<audio controls>
  <source src="horse.ogg" type="audio/ogg">
  <source src="horse.mp3" type="audio/mpeg">
  Your browser does not support the audio tag.
</audio>
<iframe width="560" height="315"
src="https://www.youtube.com/embed/k4tQGd65bqc" frameborder="0"
allow="accelerometer; autoplay; encrypted-media; gyroscope;
picture-in-picture" allowfullscreen>
</iframe>
```

Above we have a few examples of embeddable media.

First and most common, we have the image tag which, one should NOTE, has NO closing tag. Image elements contain the attribute src (source), which points to the place on the web where the image you are looking to display is hosted.

Note that if the image is hosted on the same site as the HTML page, you do not need to put in a full web address—you just need to keep the trailing address.

So for example, if your page is mypage.com and the photo you want to show is located at mypage.com/photos/photo.jpg, then you would not need to put the full address in the SRC attribute—you would only need to put “/photos/photo.jpg” because without a full address, the browser will just default to just appending the src location to your home URL.

The same goes for all the examples of media and audio files shown above.

Video and audio files have back up sources in case a browser does not support any specific file format, as well as an alternate message if a user’s browser is too old to support video and audio embeddings.

Lastly we have the iframe. The iframe is interesting because it essentially creates a window inside your window and loads another website into it. This is typically how

youtube videos are embedded into other sites. The video is not hosted on the site with a `<video></video>` tag. Instead, an iframe opens a portal to the video on youtube itself.

The iframe's **height** and **width** attributes determine the size of the window you open, and the **src** is the site that would be loaded into the iframe.

COMMON INTERACTIVE ELEMENTS

Even without JS installed into your web interface, some elements in the HTML toolkit support a level of user interaction.

The first and most powerful is the `<a>` tag used essentially to link pages together. This is basically what makes the internet such a powerful tool, and not just a collection of documents.

Links are given destinations thru the href attribute, which follows the same rules as linking images or video content:

```
<a href="/about">This is an internal link</a>
<a href="http://theknowledgehouse.org/apply">This is an external
link</a>
```

Please note for media or hyperlink locations that are external, **HTTP:// MUST BE INCLUDED IN THE ADDRESS.**

The last common element you will most likely find yourself working with is the form and input elements. Unlike everything else we have gone over, the form and input elements do not exist to give information; instead, they exist to take information, specifically user-generated information.

Form elements support many types of inputs, from checkboxes to text and long-form text, to multiple select, to multiple-choice, etc.

The last element of the form is the submit button, which would trigger the form's on action function, usually packaging the information submitted and sending it to an internal URL specified in the action attribute. This would then cause the server app to process the data and then either save it in the database or use it to specify the information to be retrieved. The label tag specifies the instructive text for either form field.

For example, google's search bar basically looks like this:

```
<form action="/api/search" method="POST">
  <label>Perform a Search: </label>
  <input type="text" id="search" name="search-term">
  <input type="submit" value="Search">
</form>
```

This creates a one text field form asking for a search term, with a submit button which displays the word “Search” as defined in the value attribute.

When the user clicks the submit button, the form would then look to the action and method.

There are two types of methods: **GET** and **POST**. GET is when we ask the server to send us something.

So for example when you go to theknowledgehouse.org, your browser is making a GET request to our website server for it to send you the HTML document for our home page.

POST, on the other hand, is when you send something to the server. In this case, we use the post method to take the user inputted search term and send it to the /API/search endpoint.

I won't go too far into what happens on the server yet, but essentially once the term gets sent to its destination, the text input would be used to generate and return a list of search results.

Below is another form—this time, a website registry form:

```
<form action="/api/register" method="POST" enctype="multipart/form-data">
  <label for="email">Enter your email:</label>
  <input type="email" id="email" name="email"><br><br>
  <label for="password">Enter your password:</label>

  <input type="password" id="password" name="password"><br><br>
  <label for="password">Confirm your password:</label>
  <input type="password" id="confirm-password" name="confirm-password"><br><br>
  <label for="img">Profile Image:</label>
```

```
<input type="file" id="img" name="img" accept="image/*">
<input type="checkbox" id="remember_pw" name="remember" value="True">
<label for="remember_pw">Remember my password.</label><br>
<input type="submit" value="Register">
</form>
```

The above form has some new things we should go over/

First, you will notice that our form element has an attribute called `enctype` that is an encryption type. Its default value is “application/x-www-form-urlencoded” .

The `enctype` attribute is used when a form uses POST as its method, and basically tells the browser how to encrypt the data before sending it away. This often includes replacing certain illegal characters, which can be used to submit malicious code into the system.

In this case, we use the value `multipart/form-data`, which is basically used to tell our form that it should be expecting a file in the submission. This was needed in this form because we are submitting a profile picture.

Next we have the email and password input types. These are basically just text inputs; however, the email field will check to ensure the submitted email is valid (contains an '@') and that the password field will show dots instead of letters as you type.

In terms of retrieving our profile pic we are using the file input type, which basically allows you to browse your computer and upload a file, and by setting the `accept` to “image/*” we are saying we will be accepting all regularly used image file formats (jpg, jpeg, png, etc)

Lastly, we have a remember me checkbox, where the initial value is set to True (which means it's checked by default).

[For more about form input types at your disposal click here.](#)

Now for the module 3 task: Fisher-Price my first website.

Make your personal website.

Now that we understand all the major parts of our website, we are going to be creating a one-page website with several sections that tell our team a bit more about you, your goals. This website will eventually become your portfolio site, but for now we are going to start with the basic about me / resume site, and then expand it from there as we begin to get a bit more advanced and do more projects.

This site will have 2 pages right now.

One page will be your homepage that will contain your header, a blurb about you and your hobbies, and a blurb about your professional goals. The end of this page will be a contact form.

The second page will basically be your digital resume.

Getting started:

Before we even start getting your site structure set up we need to set up our project directory.

As with our previous examples, we will need to use our terminal to set up the initial files and directories that you will be working with for this task.

Typically your project directory will consist of a series of HTML files and 3 folders, one for any CSS files your page uses, one for any media files or images and one for any javascript files.

We will make those folders now, but they will remain empty until later modules.

So our first set will be to CD into our module 3 directory and set up our new folder structure.

```
$ cd TKH_Modules/Module3
```

Next we will create our two HTML pages, one named index and one named resume. Please note it is a best practice to use index as your homepage file name. The others matter less, but this is the page name that most hosting

servers look for by default.

```
$ touch index.html resume.html
```

Next, let's just finish this off by setting up some empty folders for our images CSS and JS that we will be inputting later:

```
$ mkdir css js images
```

So now our module 3 folder should look like:

- Module3/
 - index.html
 - Resume.html
 - images/
 - CSS/
 - js/

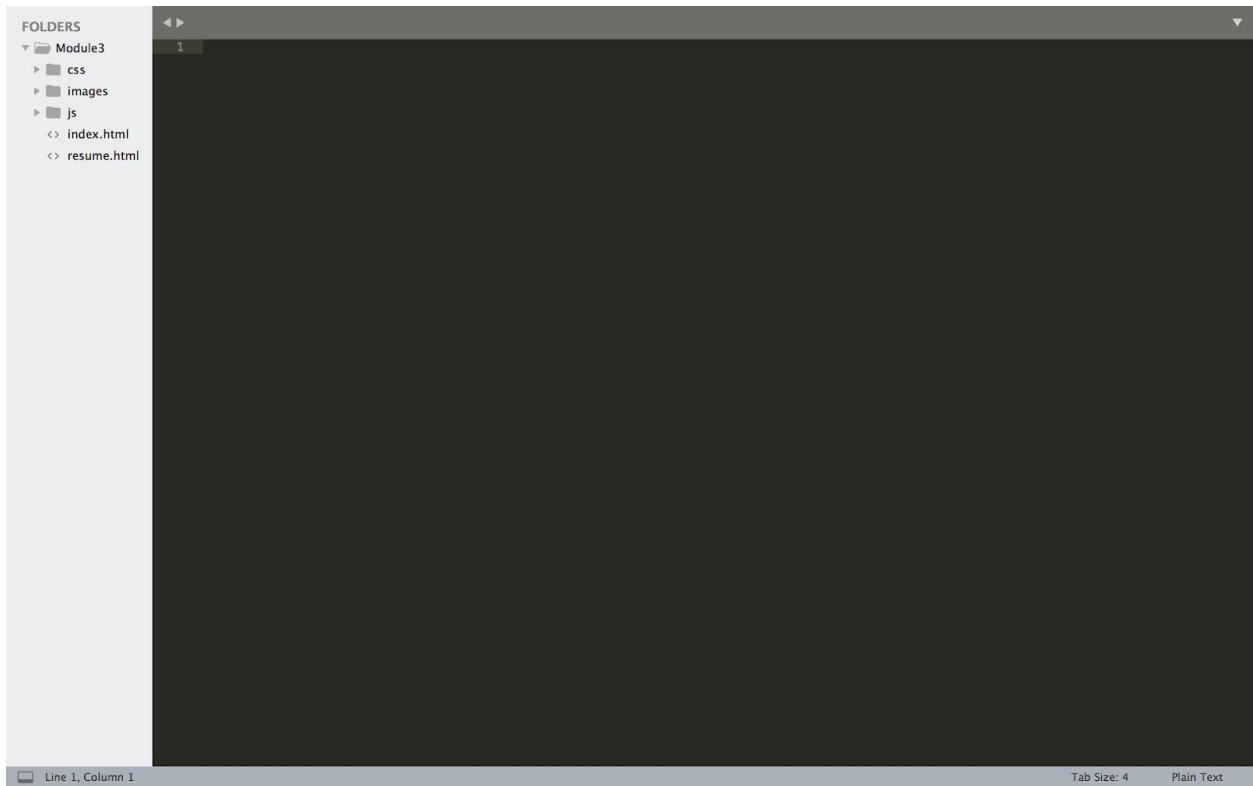
The trailing '/' just means we are representing a folder not a file.

Next we want to pick a text editor. We want to be able to open our whole directory and then go into each file and write up our code.

The one we are using is Sublime text; however, you can use whatever you want.

In Sublime go to File > Open.. and open the entire module3 folder.

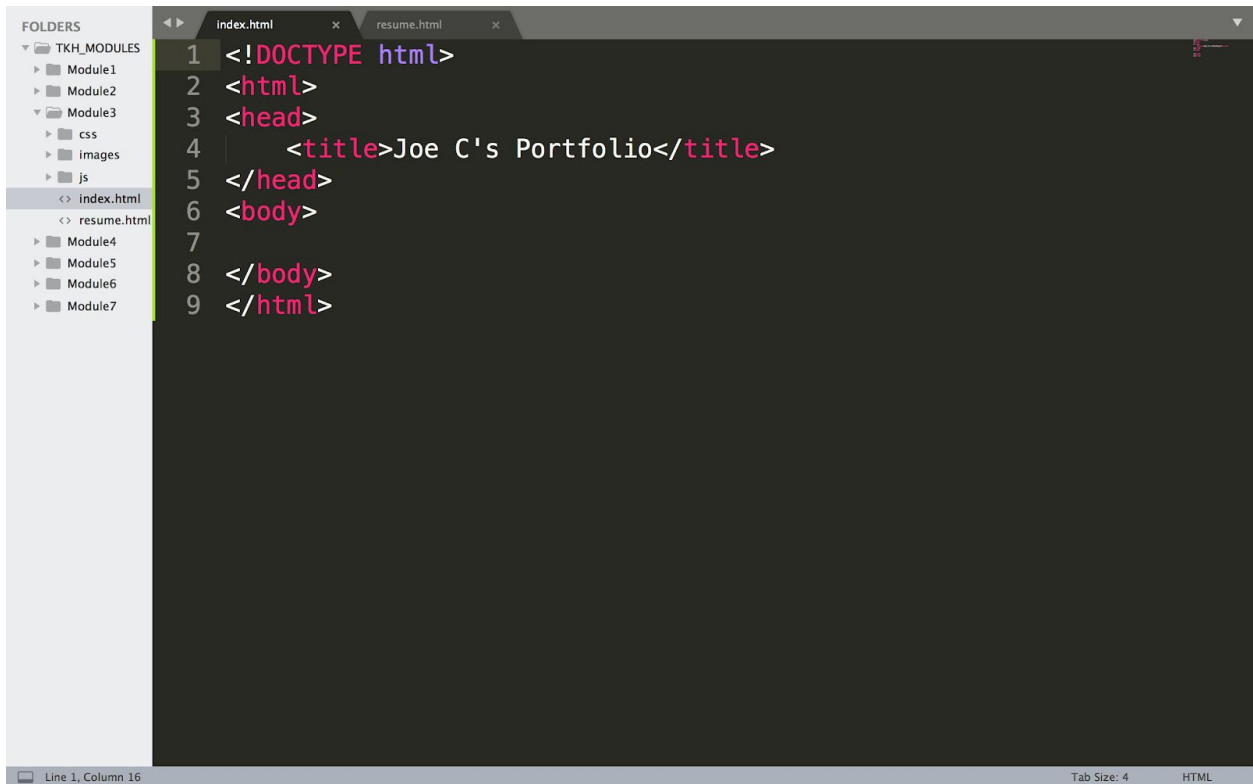
It should open looking like this:



Setting Up the Page Structure:

Before we get started, we just need to set up the main page structure on both our main page (index.html) and our resume page (resume.html)

The setup should look the same; the only difference is that for now we would have differing titles for each page:



```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Joe C's Portfolio</title>
5 </head>
6 <body>
7
8 </body>
9 </html>
```

Here the HTML tag starts and ends the page, with the Doctype tag specifying what kind of document this is.

Note the doctype tag is not necessary, but is a best practice and is needed for proper HTML validation.

In a later module, we will be adding external styles into our portfolio, but for now we are going to simply set up our content on each page, add images and link our page to both external and internal links.

First, in the about us section we are going to create several nested divs to hold our content.

So let's break down the different things we want on our index page:

1. Banner Image and Title and a menu.
2. About me section complete with profile pic, social media / GitHub links; about me text with info about goals and hobbies
3. An area about what your career goals are
4. An area listing your skillset
5. An area about your projects (with some coming soon text)

6. A contact form
7. A footer

Banner section with title and menu:

The top of your page will contain a banner section. In module 4, we'll be giving the banner a background image. In addition, this banner section will also contain a large title text and your menu for your two sections.

Since these two pieces of content live in the same container with different style factors to be applied to them, we will be creating 3 divs:

- An outer div given the class "banner"
 - An inner div given the class "title"
 - Here will go our Banner Text
 - An inner div given the class "nav"
 - Here will go the links that will be styled into a navigation bar

What your code looks like:

```
<div class="banner">
  <div class="title">
    <h1>JOE KNOWS 718</h1>
  </div>
  <div class="nav">
    **MENU ITEMS HERE**
  </div>
</div>
```

The Nav will be created using the or unordered list tag. and (ordered list) are used to create bulleted lists of items contained in or list item tags. The only difference between ordered and unordered lists is that ordered lists are bulleted with numbers and unordered lists are bulleted with dots.

A example unordered list in html will look like this:

```
<ul>
  <li>Item 1</li>
  <li>Item 2</li>
  <li>Item 3</li>
</ul>
```

And look like the following on your browser:

- Item 1
- Item 2
- Item 3

Our two items will not just be text; we will be using the `<a>` tags to create linked text.

However, we should note once again these links will be linked to the files in the home folder of your site in the href attribute.

Now we will be adding our linked menu items in the nav div:

At this point your code should look like this:

```
<html>
<head>
  <title>Joe's Portfolio</title>
</head>
<body>
  <div class="banner">
    <div class="title">
      <h1>JOE KNOWS 718</h1>
    </div>div>
    <div class="nav">
      <ul>
        <li><a href="index.html">Home</a></li>
        <li><a href="resume.html">Resume</a></li>
      </ul>
    </div>
  </div>
</body>
</html>
```

Now we want both our pages to have the same type of banner and nav, so we are just going to copy the banner div in its entirety, and paste it in resume.html, just changing the title:

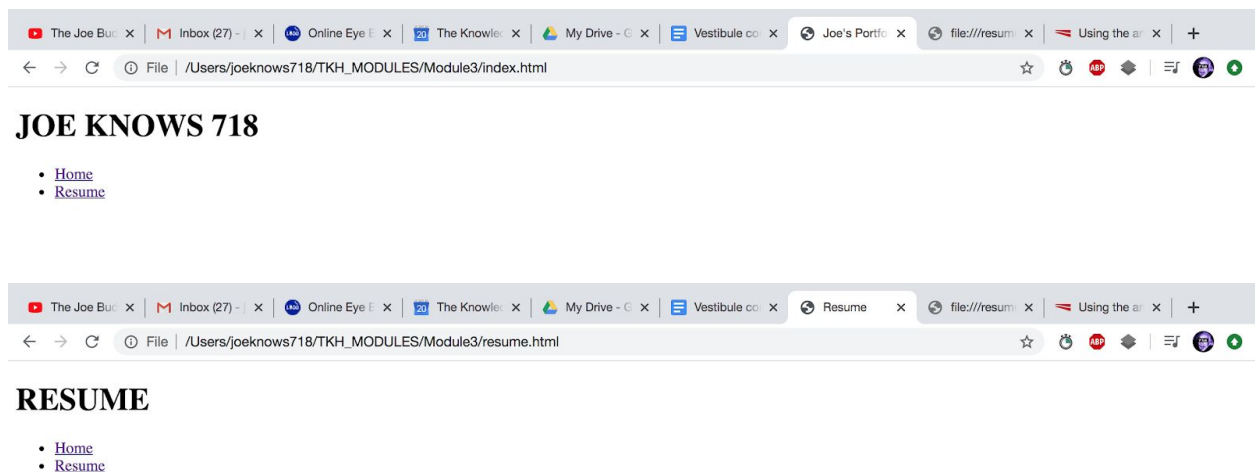
```
<html>
<head>
```

```

    <title>Resume</title>
</head>
<body>
    <div class="banner">
        <div class="title">
            <h1>Resume</h1>
        </div>
        <div class="nav">
            <ul>
                <li><a href="index.html">Home</a></li>
                <li><a href="resume.html">Resume</a></li>
            </ul>
        </div>
    </div>
</body>
</html>

```

Now, this is what your page should look like in browser:



Now that both pages banners are done, we can move on to the more page-specific content.

Next, for the home page, we are going to be adding an about me section, which will be its own div.

The div will have 2 inner divs, a profile div and an about me text div, and the profile div will also have a social media section with links to your GitHub, and LinkedIn profiles.

```
<div class="about-me">
  <div class="profile">
    <div class="profile-pic">

    </div>
    <div class="social-media">

    </div>
  </div>
  <div class="about-me-text">

  </div>
</div>
```

Now we just need to link an image and our social media in the profile section. For now, we will link them as an unordered list, but for extra style, check out the [Font Awesome library](#) to add scalable icons to our profile section.

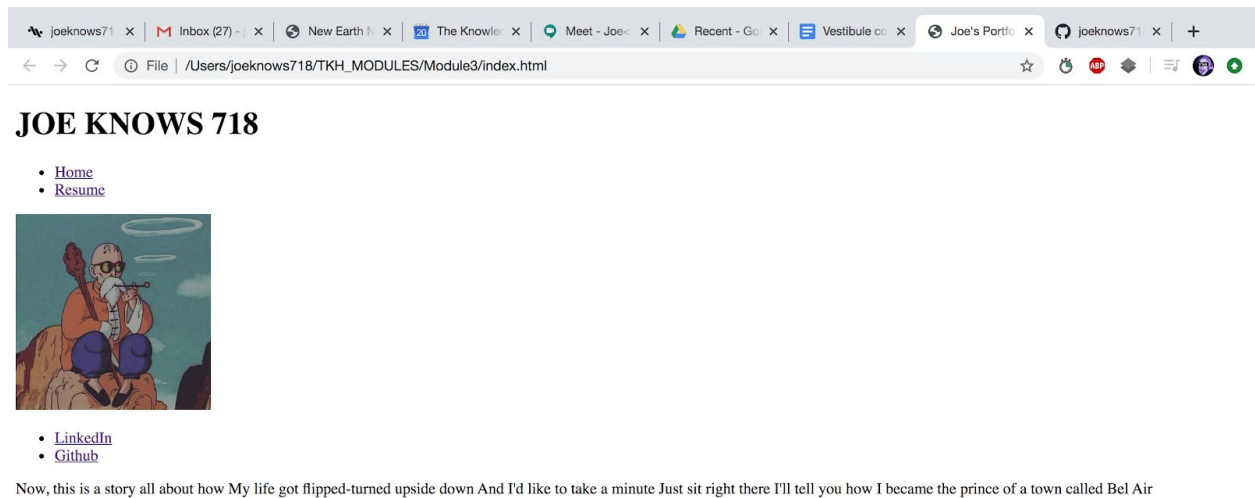
The first thing is find a pic of yourself that you want to use and just add it to your images folder in your project library. Then, simply link it in an image tag as `'/images/PIC_NAME.jpg'`. Also, just for good measure we will be adding an attribute called width to make sure our image isn't too big to handle. We are setting our width to 200px (px means pixels and is the number of pixel dots wide something is)

Next, we will be adding the social media links as linked text in an unordered list just for now until we learn how to style and add icons in module 4.

```
<div class="about-me">
  <div class="profile">
    <div class="profile-pic">
      
    </div>
    <div class="social-media">
      <ul>
        <li><a href="https://www.linkedin.com/in/joecarrano/">LinkedIn</a></li>
        <li><a href="https://github.com/joeknows718">Github</a></li>
      </ul>
    </div>
  </div>
  <div class="about-me-text">
    <p>Now, this is a story all about how My life got flipped-turned upside down And I'd like to take a minute Just sit
```

```
</div>
    right there I'll tell you how I became the prince of a town called Bel Air</p>
</div>
```

Now our homepage should look something like this:



Now that we have that done, let's move on to our next section: our career goals and skills.

Much like the last section, this will be a div with two nested divs inside, one for a description of career goals and one for a list of skills, which will be displayed as an unordered list.

```
<div class="career">
  <div class="goals">
    <h1>My Goals</h1>
    <p>My career goals are making dope things on a computer and getting paid
    accordingly.</p>
  </div>
  <div class="skills">
    <h1>My Skills</h1>
    <ul>
      <li>HTML&CSS</li>
      <li>PYTHON and FLASK/Django</li>
      <li>JS/React/Node</li>
      <li>Handstyles and Throw-Ups</li>
      <li>Herding Cats</li>
    </ul>
  </div>
```

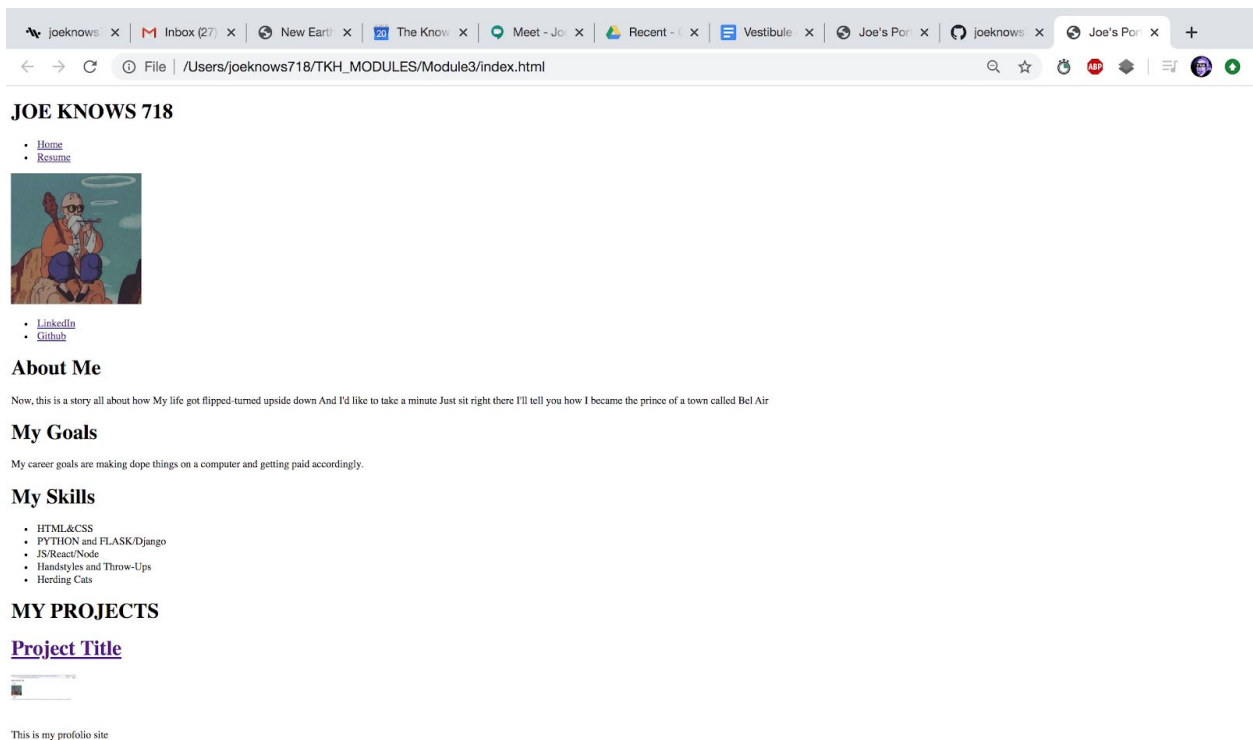
Next, we will be adding out the project section and contact form. The project section will

contain a project title that is linked to the project, screenshot, and description.

In this example our first project placeholder will be our actual portfolio (please note a screenshot of the project was added into our image folder saved as screenshot.png:

```
<div class="projects">
  <h1>MY PROJECTS</h1>
  <div class="project">
    <a href="index.html"><h1>Project Title</h1></a>
    
    <p>This is my portfolio site</p>
  </div>
</div>
```

Now our site looks like this:



You may notice everything is just in one giant column. Eventually, we are going to reorganize all this content with CSS; right now, the goal is just to set up the content. We will stylize this in module 4.

The last 2 things we need now are a contact form and a footer.

First, let's create our form. We want a user to give us their name, email, and submit a message. Right now, this form is just for show; later we can use it to send an email message to yourself when someone fills the form out, once we learn how to develop servers.

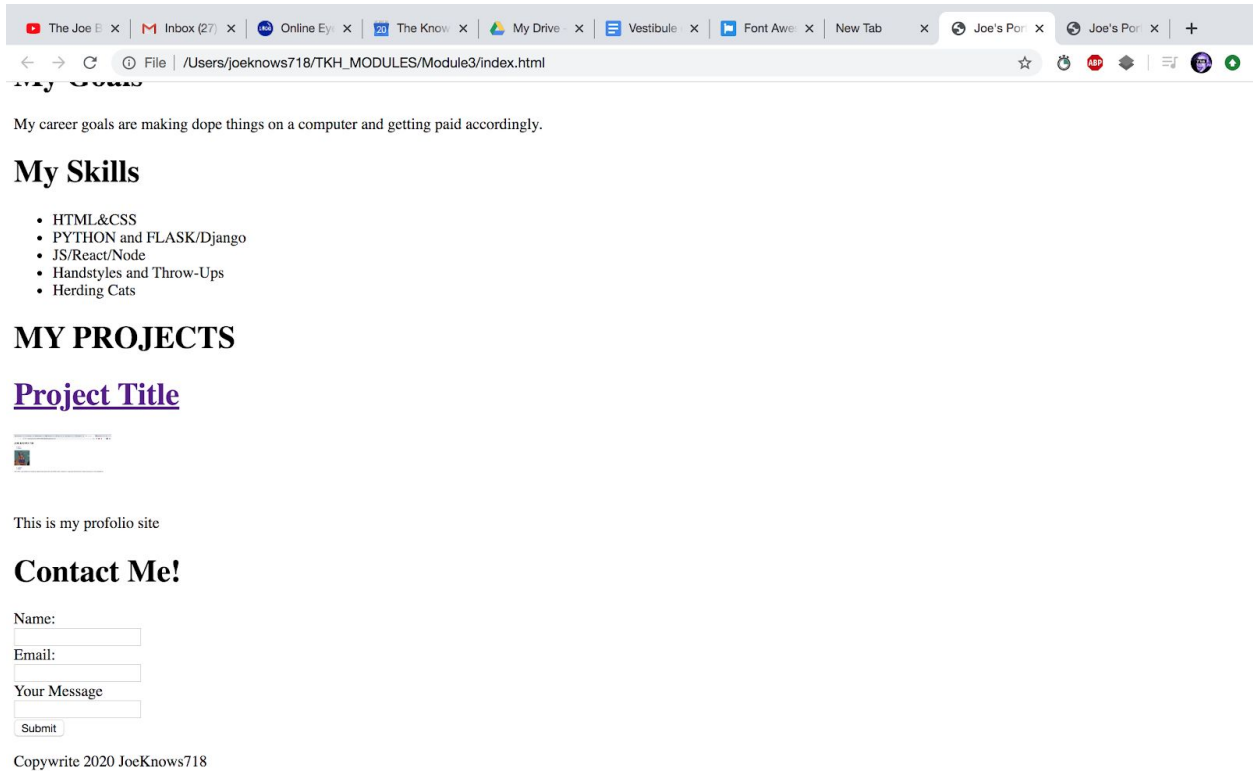
Basically we will have a form with 4 inputs: 3 text inputs and a submit input. This form will be housed in the div classed as contact-form. As you can see, we also added labels for each form. The for attribute in the label tag allows the browser to know which label corresponds with each input:

```
<div class="contact-form">
  <h1>Contact Me!</h1>
  <form class="contact">
    <label for="name">Name:</label><br>
    <input type="text" name="name"><br>
    <label for="email">Email:</label><br>
    <input type="text" name="email"><br>
    <label for="message">Your Message</label><br>
    <input type="text" name="message"><br>
    <input type="submit" name="Contact">
  </form>
</div>
```

Our last step will be adding a footer: this is just going to be a div classed as footer, with some text.

```
<div class="footer">
  <p>Copyright 2020 JoeKnows718</p>
</div>
```

And here is our final homepage:



Now we before we move on to our resume page (which is much easier), let's just copy over our footer since that will appear on both pages:

Our starting code after all the copy and pastes for the resume page should look like:

```
<body>
  <div class="banner">
    <div class="title">
      <h1>RESUME</h1>
    </div>
    <div class="nav">
      <ul>
        <li><a href="index.html">Home</a></li>
        <li><a href="resume.html">Resume</a></li>
      </ul>
    </div>
  </div>
  <div>

    <!--STUFF GOES HERE -->
```



```
<div class="footer">
  <p>Copyright 2020 JoeKnows718</p>
</div>
</body>
</html>
```

For our resume, we just want to create a timeline of our experiences. Each job entry will have its own section containing the following:

- A title and a company
- A start and end date
- A small description and list of duties

Lastly, you will also have a section built for education as well.

Your code should look like this:

```
<div class="jobs">

</div>
<div class="education">
  <h1>CUNY BARUCH</h1>
  <p>Majored in Business Admin</p>
</div>
```

Now in our Jobs div, we will create a structure for each job entry that we outlined above, including title, length, description, and duties/accomplishments.

In the below code I have added two separate jobs, each one having the same content types and organization.

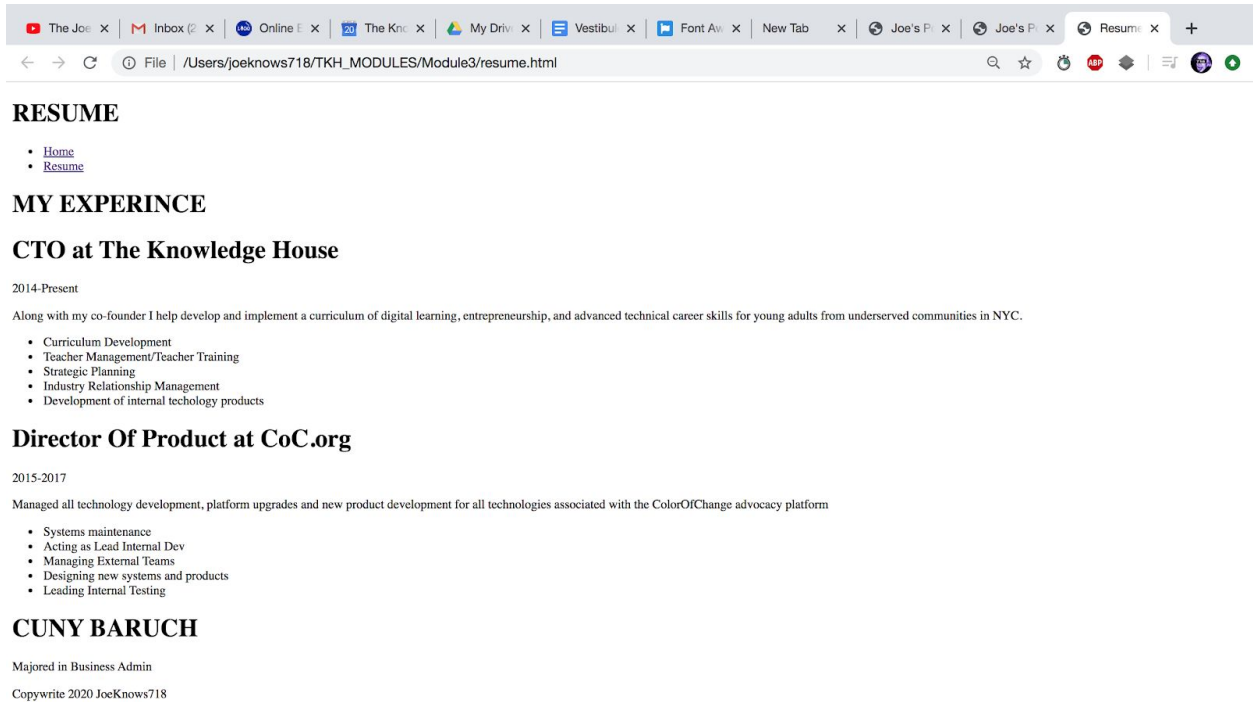
```
<div class="jobs">
  <h1>MY EXPERIENCE</h1>
  <div class="job">
    <h2>CTO at The Knowledge House</h2>
```

```

        <p class="date">2014-Present</p>
        <p class="desc">Along with my co-founder, I help
develop and implement a curriculum of digital learning,
entrepreneurship, and advanced technical career skills for young
adults from underserved communities in NYC.</p>
        <ul class="duties">
            <li>Curriculum Development</li>
            <li>Teacher Management/Teacher Training</li>
            <li>Strategic Planning</li>
            <li>Industry Relationship Management</li>
            <li>Development of internal technology
products</li>
        </ul>
    </div>
    <div class="job">
        <h2>Director Of Product at CoC.org</h2>
        <p class="date">2015-2017</p>
        <p class="desc">Managed all technology development,
platform upgrades and new product development for all technologies
associated with the ColorOfChange advocacy platform</p>
        <ul class="duties">
            <li>Systems maintenance</li>
            <li>Acting as Lead Internal Dev</li>
            <li>Managing External Teams</li>
            <li>Designing new systems and products</li>
            <li>Leading Internal Testing</li>
        </ul>
    </div>
</div>

```

Now we should be done and have a resume page which looks something like this(with your experience of course:



Submitting Module 3:

You will be submitting your HTML only site through Github.

Go into the module 3 folder on your master branch, and inside this folder add another folder called 'portfolio'.

Module_3

Portfolio/

- Index.html
- resume.html
- images/
 - Image1.jpg *This is a placeholder for your image files

This must be submitted through Github.