**HTML:**

* Hyper Text Markup Language
* For a website, HTML is used to give the fundamental structure! (CSS provides styling and JavaScript is responsible for functionality!)

**ELEMENTS:**

* .html pages are made up of elements.
* An element usually begins and ends with opening and closing tags respectively, and enclose content within these tags.
  1. These are **container elements.** For example:

<h1> This is my first webpage </h1>

* 1. Some are **standalone elements** (Don’t hold content within their tags). For example, given **below is the first line that every html file begins** with:

<!DOCTYPE html>

**HEADER ELEMENT:**

* <h1></h1> refers to the **header element** for headings.
* Similarly, h2 also refers to a heading – but smaller. With a larger number after the ‘h’, heading size goes down!

**PARAGRAPH ELEMENT:**

* The <p></p> element refers to the **paragraph element** for content.

**COMMENT:**

* <!—represents the opening tag for comments.
* --> represents closing tag.

For example:

<!--TODO: Make link to dog pictures!-->

**MAIN ELEMENT:**

* All page content(like comments, paragraphs, etc.) excluding headers can be wrapped within the <m> and </m> tags that correspond to the **main element**.

**NESTING:**

* Nesting is when you wrap elements within other elements.
* For example, the paragraph element is nested within the main element, which is itself nested within the header element. Similarly, the header element is nested within the body element, which is itself nested within the html element (<html></html)

**IMAGE ELEMENT:**

* Denoted by <img> opening tags; no closing tags.
* All content enclosed within the opening tag itself.
* **attributes** and **values** come in here.
* The **attribute-value** pair specify the behavior of the particular element.

For example:

<img src=“https://as2.ftcdn.net/v2/jpg/02/81/74/99/1000\_F\_281749923\_YiMgJI7Y7ngE6FqUSr0eEWdaeZ3UB7EZ.jpg>

* Image elements are also defined by the **alt** attribute. The value for this attribute corresponds to a simple text description of the image in question.
* So, if the image fails to load for any reason, web browsers will display this text instead.

For example:

<img src=“ <https://as2.ftcdn.net/v2/jpg/02/81/74/99/1000_F_281749923_YiMgJI7Y7ngE6FqUSr0eEWdaeZ3UB7EZ.jpg>” alt=“Cool dog resting on an adjustable beach chair while sipping drinks”>

**ANCHOR ELEMENT:**

* Represented by <a> </a> (has opening + closing tags)
* Used to add hyperlinks to text, to redirect to sites.
* Defined by attribute **href**, with the value being the URL to the respective website.

For example:

<a href= “https://dogs.com”>Click here for dog pics</a>

* Note here that, like before, the attribute-value pair lies within the opening tag!
* Small nuance: The text you want hyperlinked should be enclosed normally b/w opening and closing tags.
* Additionally, the anchor element is defined by the **target** attribute, whose value can be set to **\_blank** . This makes the hyperlink **open in a new tab** instead of the same.

For example:

<a href= “https://dogs.com” target= “\_blank”>Click here for dog pics</a>

* **Placeholder** is when you insert a hash(#) sign as the value for the **href** attribute – in place of the actual site URL.
* This is frequently done when working with JavaScript, as placeholder ensures the ‘hyperlink’ functionality still works while the JavaScript team works on building the website in question.
* Until then, the placeholder redirects user to the same .html page!

**UNORDERED LISTS:**

* <u></u> refers to **unordered list** element.
* These elements are usually paired with the <li></li> elements (list elements) that correspond to individual bullet points.

**NOTE:**

* Don’t nest header elements around any text that you don’t want having the header style (bold).
* For example, if you want an unorded list element under a heading, but don’t want the header element’s bold style to be applied to the unordered list (and its list elements), **ensure to nest the header element around the <ul> </ul>** element.

**ORDERED LISTS:**

* Ordered lists are numbered, unlike unordered lists (which are bulleted)
* <ol> </ol> represented ordered list element.
* The same list element is used here too, though. So <li> </li> is here too.

**STRONG ELEMENT:**

* <strong> </strong> refers to strong element.
* These are container elements (Content enclosed b/w opening and closing tags)
* Used for phrases of ‘strong importance’.
* Makes content **bold**

For example:

<p> Things dogs <strong>hate</strong> </p>

**EMPHASIS ELEMENT:**

* <em> </em> refers to emphasis element.
* Also a container element.
* Used for ‘emphasising’ certain phrases.
* Makes content *italic*

For example:

<p> Things dogs <em>love</em></p>

# FORM ELEMENTS:

**FORM ELEMENT:**

* <form> </form> initiates the form design!
* **action** attribute for this element defines where this form links/redirects to. It could be a different webpage for example.
* This <form> element nests around all other features of the form (button element, input element, etc.) that we’ll see below.

**For example:**

<form action= ‘https://blahblahblah.com’> </form>

**INPUT ELEMENT:**

* <input> refers to input element. (Has no closing tags, because it is not a container element. Why is it not a container element? Because some <input> types like text cannot ‘pre-contain’ any text, and only waits for user input.)
* Has **type** attribute that defines the type of input element (values for this attribute can be text, radio, checkbox, etc.)
* Plus, the **id** attribute is the unique identifier for this input element. This is used by web browsers to identify elements uniquely.
* **Placeholder** attribute gives a hint, in the form of a text value, to the user on what to enter in the input element.
* **Required** attribute has no values, and is just used to make it mandatory for the user to fill this <input> before moving forward.

For example:

<form action= ‘https://blahblah.com’>

<input type= ‘text’ id= ‘indoororoutdoor’ placeholder= ‘Indoor/Outdoor?’ required>

</form>

* Again, note that the <form> element **nests around** everything else!
* To create radio buttons, use type= “radio” for <input> element!
* A normal set of 2 radio buttons would only allow you to select one of them. Selecting the other would automatically de-select your previous selection.
* To implement this functionality, you need a **radio group**. A radio group is a set of radio buttons that follow this functionality.
* To create a radio group, just ensure to name your desired set of radio buttons with the same **name** attribute!

**BUTTON ELEMENT:**

* <button> </button> is a container element with the text you want displayed on the button being in between the opening and closing tags.
* Has attribute **type** to define what kind of button you’re designing (‘submit’ is an example of a value for this attribute.)

For example:

<form action= ‘bksnfdmvlc.com’>

<input type= “” id= “” required= “” placeholder = “”>

<button type= “submit”> SUBMIT to know your dog’s personality! </button>

</form>

**LABEL ELEMENT:**

* <label> </label> element gives a sort-of ‘text label’ for every input element! It **nests around** every input element!
* It appears to the left of the input field on the .html – describing what should go into the input box to the user.
* Has a **for** attribute that should evaluate to the **id** value of the corresponding input element!
* And, as usual, it’s a container element with the necessary text going in b/w the tags.

For example:

<form action= ‘bksnfdmvlc.com’>

<label for= “indooroutdoor”> Is yours an indoor/Outdoor dog?

<input type= “” id= “indooroutdoor” required= “” placeholder = “”>

</label>

<button type= “submit”> SUBMIT to know your dog’s personality! </button>

</form>

**IMPORTANCE OF <LABEL> ELEMENT:**

* This can be overlooked as a sort-of menial element, but it is not!
* THIS IS CRUCIAL FOR ADDRESSING ACCESSIBILITY NEEDS.
* ONE, people with visual impairments and cognitive disabilities rely on **screen readers** that ‘read aloud’ the webpage’s text.
* By attaching a <label> to every <input>, the screen reader ensures to read out the ‘label’ to these disabled users – giving them an idea of what to input into the <input>.
* TWO, the <label> element – with its **for** attribute linking to the **id** of the <input> -- redirects the average user(not talking about impaired users) to the corresponding <input> box if clicked. Yes, by just clicking on the <label> text on webpage, your cursor will redirect to the corresponding input field.
* This is particularly useful when there are a lot of input fields on a webpage. Users, then, can simply rely on clicking on the ‘labels’, as that would redirect them to the appropriate <input> box.

**CREATING CHECKBOXES:**

* <input> elements of **type= “checkbox”** produce checkboxes.
* This <input> element carries a lot of the same attributes (id, type, name, etc.), but some new attributes pop up based on the type of <input>
* For type= “checkbox”, a new attribute **checked** (with no corresponding value) is available, which sets the default state of the checkbox to a checked one.

For example:

<label for= “obedient”>

<input type= “checkbox” id= “obedient” checked name= “personality”> Obedient?

</label>

**DIV ELEMENT:**

* Abbreviates to ‘division’ element (One of the most commonly used .html elements)
* It’s an all-purpose container that contains other elements.
* Creates a division for all the elements contained within it, and is usually used along with CSS to style the desired section!
* Has a **class** attribute that can be valuated to any preferred name by the user. Once a class attribute is defined, you can set a style(with CSS) for the class, and every element nested within the <div> is applied with that style!

**FOOTER ELEMENT:**

* This comes outside the <main> </main>, but nested within the <body> </body>
* <footer> </footer> refers to the footer element.
* Now, this element is usually just created for **screen readers** & **SEO(Search Engine Optimisation)** purposes.
* You can add a <p> </p> paragraph tag within the <footer> </footer> element, and add desired text you’d like to display on your webpage’s footer.
* Plus, since anything you add to the footer should look separate from the main contents of the page, you can wrap the above <p> paragraph element inside a <small> </small> element.

For example:

<footer>

<small><p> NO COPYRIGHT – <a href= <https://pexels.com> target= “\_blank”>Pexels.com</a> </p> </small>

</footer>

**HEAD ELEMENT**:

* <head> </head> represents head element.
* Again, these things rarely show up on webpage.
* This element is nested outside the <body> </body> element, but within the <html> </html> element.

**TITLE ELEMENT:**

* <title> </title> represents the title element.
* You can nest this element within the <head> element, and you’ll see that the webpage title will change to the text input that you contain within the <title> </title> element.

For example:

<!DOCTYPE html>

<html lang= “en”>

<head>

<title> 16 Personalities for dogs! </title>

</head>

</html>

**IMPORTANT CONCEPTS:**

* 1. Elements of HTML5 (like ‘body’, ‘main’, ‘header’, ‘paragraph’, etc.) are useful because they help in **accessibility**(screen readers, perhaps) and **search engine optimization (SEO)**.
  2. The <body> element is what houses ALL CONTENT of webpage (apart from the auxiliary info. stored by the <head> and <footer> tags. So, yes, <main> would be nested within the <body> element – just like most every HTML5 element.
  3. <main> element is designed to store ‘unique data’ that isn’t present anywhere else in the HTML document (<head>, <footer>, etc.). So, for example, this would be considered redundant code:

<!DOCTYPE html>

<html>

<head> Sample webpage </head>

<body>

<h1> This is my first webpage!</h1>

<main>This is my first webpage!</main>

</body>

<footer>This is my first webpage! </footer>

</html>

This is highly redundant code, with the main element containing information duplicated in the <head> and <footer> tags outside it. Designing such code (where <main> doesn’t house unique data) would cause:

* **Accessibility problems**
* **SEO problems** (when web browsers encounter such duplicate information on a HTML document, the search engine punishes the .HTML doc with a low rank, affecting the site’s visibility and revenue.

4) Nesting is a ‘readability’ practice by web developers. It’s so much more pleasing to the eyes, and easier to skim through!

5) <section> element (new learning!) is used to partition webpage content into ‘sections’ that improve the webpage’s SEO ranking and readability by web browsers! You can nest a <section> </section> around your desired content.

6) <figure> </figure> element (new learning) is usually nested around an <img> element to act as **captions**. The <figure> element is actually a **container element** that shall have another element (<figcaption> </figcaption>) nested within it.

7) <figcaption> </figcaption> stores the image captions between the tags.

8) void elements are those elements that don’t need closing tags.

9)**key concept**: how you write code(with or without breaks) on the HTML code editor has no impact on the actually visible line breaks on the webpage. The line breaks on the webpage depend on the type of element. If the element is a **block-level element**(like <div>, <h1>, <p>, <head>, <footer>), it will trigger a line break on the webpage. However, other elements (called **inline elements**) like <em>, <strong>, <a>, <img> will not prompt the web browser to add line breaks! The flow is maintained with these **inline** elements. You can, however, force a visual line-break between **inline elements** with a <break> element wherever desired.

10) **void elements** have only start tags, but no end tags. (e.g: <input>)

11) learned **value** attribute to <input> element. This is important, because when you submit a form with a particular radio button selection, (in the back-end), what is seen is ‘the radio button of **name**= “indoor-outdoor” and **value**= “\_\_\_\_” is selected.’ Without providing an explicit **value** attribute, its hard to identify the button that was selected (as all radio buttons of the same radio group will have the same **name** attribute)

12) if you want to separate a radio group from another radio group, instead of forcing a break between the two **inline <input> elements**, nest the two radio groups within different <fieldset> **block-level elements**. This way, the <fieldset> block-level element not only prompts the browser to create a line-break, but also provides ‘structure’ to the webpage (useful for accessibility and SEO purposes).

13) also, nest a <legend> element within the <fieldset> element, in order to give the user an idea of what to expect from this particular part of the form, or this particular **radio group** in this case.

14) any data you want visible on the webpage should go into the <body> element. Other metadata that you don’t want to be visible should go outside it, either in the <head> or <footer> elements. Here’s more information on that:

A screenshot of a computer

Description automatically generated

15) <!DOCTYPE html> is a special string called a **declaration** that always goes on top of the html document. It tells the browser that it’s currently reading a HTML5 document.

16) <head> has a <meta> element nested within it (usually), with the attribute **charset= “utf-8”** – which, as you might’ve deduced, tells the browser how to encode the page.

For example:

<!DOCTYPE html>

<html>

<head>

<meta charset= “utf-8”>

</head>

<body>

<main>

</main>

</body>

<footer>

</footer>

</html>

**IMPORTANT SECURITY FEATURE:**

* The <a> anchor element has an attribute called **rel**, which abbreviates to relationship. By equating this attribute to a value of ‘noreferrer noopener’, you improve security of your webpage.
* How? **noreferrer** and **noopener** means that link you opened cannot access the current html page you’re in because of the **‘noopener’** value and the ‘**noreferrer’** value provides additional security by not allowing the linked page’s owner to see the ‘referrer’ who visited their page.

**ALWAYS ENABLE BRANCH PROTECTION RULES ON GITHUB:**

* Main 2 rules to enable are:
  1. Require pull requests to be made
  2. Require signing before commits.
* Protect the base ‘main’ branch with the above ruleset.
* Then, any commits you make, ensure to use a separate branch from the main on your local, and then sign it and push this new branch to GitHub.
* Then, a pull request is created on GitHub. Review and approve pull request.

WHAT THE HELL? WHY IS SIGNING A COMMIT WITH SSH KEY SO FKN HARD?

Mannnn, fk