INTRODUCTION to HTML and CSS

HTML BASICS:

HTML is the acronym for Hyper Text Markup Language, which is the coding language used to define the contents of documents that will be displayed in web browsers. HTML is also used to describe the type of document.

HTML tags are used to declare elements in an HTML document. An HTML tag is the portion of an element that contains the name of the element, and defines the location (beginning and end) of the element. Within the tag will be the contents and attributes of the element. Tags must follow HTML syntax rules in order to properly display the elements.

• Most elements have three primary parts: the opening tag (the element name, enclosed in angle brackets: `<element>`), the content (varies, can be: text, attributes, etc.), and the closing tag (the element name preceded by a forward slash /, enclosed in angle brackets: `</element>`).

  **Example:** `<p> This is a paragraph element. </p>`
  In this example the *opening tag* `<p>` defines the beginning of a paragraph element, the *closing tag* `</p>` defines the end of the paragraph element, and the text is the *content*.

• HTML syntax rules require all tags to be contained within a pair of angle brackets, the `<` (less than) and `>` (greater than) characters.

  **Example:** `<element>` (an element name, between a pair of angle brackets)

• All *opening* tags are enclosed in `<` (less than) and `>` (greater than) characters.
  
  o `<element>`
  
  o Opening tags may include additional attribute information before the `>`

    **Example:** An element with a *class* assigned `<element class="xxx">`
• All closing tags are enclosed in < (less than) and > (greater than) characters with a / (forward slash) preceding the element name.
  o </element>
  o A closing tag has no additional information, it simply tells the browser this is the end of the element.
    
    Example: <element> (content) </element>

• Some elements are self-closing. They require contents, but do not require a closing tag. One of these is the image element <img>.
  
  Example: <img src="path/file.jpeg">
  In this example, the beginning of the image element is declared by the tag <img followed by a src attribute pointing to the image source (src="path/file.jpeg") followed by the closing > (greater than) character. No closing tag is required.

• A few HTML tags are “stand-alone” and do not require any contents or closing tag.
  Example: <br> (this tag defines a line-break element, like the “enter” key on a keyboard)

• It is possible to nest one, or more, tags within another tag (parent / child). The syntax rules require a child tag to close before its’ parent tag closes. (More on element nesting in the HTML Document Tree section, below.)
  Example: <parent> <child> </child> </parent> (correct syntax)
  Example: <parent> <child> </parent> </child> (incorrect syntax)

• Comments, which are not displayed when viewing an HTML document in a browser, have a special syntax. Comments begin with <!-- and end with --> and the content between is considered the comment.
  Example: <!-- Comment text goes here. -->

• The following is an example of a short segment of HTML code that includes comments (begin BANNER and end BANNER), a parent tag (element = div), a self-closing tag (element = img), and a nested child tag (element = p).
  
  <!-- begin BANNER -->
  <div>
    <img src="site_images/logo.jpg">
    <p> This is my Site Logo. </p>
  </div>
  <!-- end BANNER -->
THE HTML DOCUMENT TREE:
HTML coding is organized in a hierarchical “tree” structure, like a family tree. There are ancestors, descendants, parents, children, and siblings. *(Note: The HTML Document Tree structure directly impacts how CSS rules are applied.)*

The example HTML below will be used to illustrate the HTML Document Tree structure.

```html
<body>
  <div id="content">
    <h1>Heading here</h1>
    <p>Some text.</p>
    <p>More <em>italic</em> text.</p>
    <hr>
  </div>
  <div id="nav">
    <ul>
      <li>item 1</li>
      <li>item 2</li>
      <li>item 3</li>
    </ul>
  </div>
</body>
```

Here is a diagram of the example HTML Document Tree structure.
Ancestor

An *ancestor* refers to any element that is connected, but is further up the document tree - no matter how many levels higher. In the diagram below, the `<body>` element is the *ancestor* of all other elements, it resides at the top of the tree.

![Ancestor Diagram](image)

Descendant

A *descendant* refers to any element that is connected, but is lower down the document tree - no matter how many levels lower. In the diagram below, all elements that are connected below the `<div>` element are *descendants* of that `<div>` element.

![Descendant Diagram](image)

Parent and Child

A *parent* is an element that is directly above, and connected to, another element in the document tree. In the diagram below, the `<div>` element is a *parent* to the `<ul>` element.

A *child* is an element that is directly below, and connected to, another element in the document tree. In the diagram above, the `<ul>` element is a *child* to the `<div>` element.

![Parent and Child Diagram](image)
Sibling
A sibling is an element that shares the same parent with another element. In the diagram below, all `<li>` elements are siblings because they all share the same parent, the `<ul>` element.

![Diagram showing sibling relationship]

COMMON HTML ELEMENTS:
The following are HTML elements that are used in nearly every HTML page:

- `<!DOCTYPE HTML>`
  DOCTYPE is not technically an element. The tag is a declaration, at the beginning of every HTML document, that tells the browser that the document is HTML (and sometimes what version of HTML). This declaration does not have a closing tag and is not case sensitive. For HTML5, you can use `<!DOCTYPE html>` or `<!doctype html>`

- `<html> </html>`
  The `<html>` and `</html>` tags declare the beginning of the HTML element and the end of the HTML element, respectively. The opening tag of the HTML element begins immediately after the `<DOCTYPE>` tag and the closing tag is the last line of the page. Every other element in the page will be nested inside the HTML element opening and closing tags.

- `<head> </head>`
  The head section of an HTML document is declared by the `<head>` tag. The head section contains other nested tags that include specifics about the web page, such as the page’s title and author, links to external CSS or JavaScript files, metadata such as keywords, and much more. The contents inside the head tag are not displayed on the web page, but can have a significant impact on how the rest of the elements are displayed.

- `<title> </title>`
  The `<title>` tag declares the title element of the web page. The contents of the `<title>` tag will appear on the title bar, or the tab, in the browser window and is one of the first items found when the public conducts a web-based search.
• **<body> </body>**
The **<body>** tag declares the portion of the page that contains the primary content intended to be *visible* when the page is viewed in a browser. Typically, any elements that are not nested within the **<body>** tag will not be displayed by a browser.

• **<h1> </h1> ... <h6> </h6>**
The heading tags, **<h1>** through **<h6>**, are used to declare section headings, much like title bars at the beginning of book chapters. There can be up to 6 heading tags in an HTML document, typically encountered in sequence (1, 2, 3...) as the page progresses. The attributes for each can be defined using CSS (more on this later).

• **<p> </p>**
A **<p>** tag is used when you want to structure content into a paragraph. An HTML document can have as many **<p>** tags as necessary for your intended page structure.

• **<b> </b> or **<strong> </strong>**
The **<b>** tag and **<strong>** tag will format the enclosed text content as **bold**. Although a browser will display either in the same way, **<b>** is recommended to be used when the intention is for visual effect only (simply bold text) while **<strong>** is recommended to imply extra importance.

• **<i> </i> or **<em> </em>**
The **<i>** tag and **<em>** tag will format the enclosed text content as **italic**. Although a browser will display either in the same way, **<i>** is recommended to be used when the intention is for visual effect only (simply italic text) while **<em>** is recommended to imply extra emphasis.

• **<u> </u>**
The **<u>** tag will format the enclosed text content as **underlined**.

• **<span> </span>**
The **<span>** tag is a generic **inline** container that can be used to apply attributes to a section of HTML code. The attributes are defined using CSS and will be applied to anything contained within the **<span> </span>** tag. You can nest a **<span>** tag within any other tag, including another **<span>** tag. You can also nest any other tag within a **<span>** tag.

• **<div> </div>**
The **<div>** tag (division tag) declares a container used to group portions of code, allowing you to apply CSS styling attributes to the content within the **<div> </div>** tag.

• **<ol> </ol> or **<ul> </ul>** and **<li> </li>**
The **<ol>** tag declares an **ordered list**, which will display list items hierarchically, with each item preceded by a **number**. The **<ul>** tag declares an **unordered list**, which will display list
items hierarchically, with each item preceded by a bullet. For both, the items in the list are declared using the `<li>` tag, and both must include at least one item.

- `<img src=""/>`
  The `<img>` tag declares an image element at the tag location. The `<img>` tag is self-closing, but it requires, at minimum, that the image source (`src=""`) be included (see example, above).

- `<a href="" /> </a>`
  The `<a>` tag declares an anchored link element at the location of the tag. An anchored link can be a pointer to a location within the current document (a paragraph farther down the page, for example), another page on the current web site, or another web site, and more. The `<a>` tag requires, at minimum, that the referenced item and location (`href=""`) be included within the tag.

Reference information on HTML Elements: https://www.w3schools.com/tags/ref_byfunc.asp

**CSS BASICS:**

**CSS** is an acronym for Cascading Style Sheet, which is a method to define rules that influence how HTML elements are displayed, and when certain actions will occur. CSS is also used as the name for the markup language that defines these rules and actions.

- CSS rules have two required parts – **Selector** and **Declaration**:
  1. **Selector**
     A CSS selector can target an HTML element, a class, or an ID.
     - **HTML element selectors** define which HTML element will be influenced by the rules defined in the declaration portion (see below). Rules defined for an HTML element will be applied to every instance of that element.
       - **Example:** `h1 {color: blue;}`
         Here the selector targets the `h1` element and the declaration sets text color to blue, which will be applied to every `h1` element in the HTML document.

     - **Class selectors** define a class, by name, that can be assigned to any HTML element. Class selectors begin with a dot (.) followed by a unique class name.
       - **Example:** `.blue {color: blue;}`
         Here the selector targets a class named `blue` and the declaration sets text color to blue, applied to any element in that has the class assigned.
- **ID selectors** define an identifier, by name, that can only be applied to a single HTML element. A CSS ID is used to identify a particular HTML element, allowing it to be specifically targeted within an HTML document. CSS ID selectors begin with a hash tag (#) followed by a unique ID name.

  **Example:** `#first {font: bold;}`

Here the selector targets an ID named `first` and the declaration sets the font to be displayed **bold**, applied only to the specific element assigned the ID.

2. **Declaration**

- A **CSS declaration** is where the style properties are declared that will apply to the target selector. A CSS declaration is comprised of a **property name** and the **value** assigned to that property. Both are enclosed in **curly braces** `{ }` with the property name and value separated by a **colon** `:`), and the declaration terminated by a **semi-colon** `;`.

  **Example:** `h1 {color: blue;}

Here the declaration **property name** is **color** and the **value** is **blue**. This example sets all instances of the `h1` element to be displayed in **blue** text.

### Methods to define CSS rules:

1. CSS rules can be defined in separate documents, called **External Style Sheets**, that are then referenced in an HTML file. The reference is accomplished by declaring an HTML **link** element, which points to the External Style Sheet, in the HEAD section of the HTML.

   **Example:** `<link href="styles.css" rel="stylesheet">`

   In this example a **link** element is declared which references an External Style Sheet named `styles.css` (**href**="**style.css**"). The relationship statement indicates that the referenced document is a **stylesheet** (**rel**="**stylesheet**"). Note that link elements are self-closing (do not have a closing tag).

2. CSS rules defined in the HEAD section of HTML files are called **Internal Style Sheets**.

   **Example:** `<head>  
   <style> body {background-color: gray;} </style>  
   </head>`

   In this example the **HTML element selector** is **body** and the **declaration** sets the **property** **background-color** to the **value** **gray**, resulting in the browser displaying the page with a **gray** background.

3. CSS **style declaration** rules that are defined within the tag of an HTML element are referred to as an **Inline Styles**.

   **Example:** `<h1 style="color:blue;"> A Blue Heading. </h1>`

   Here the **HTML element** is `h1` and the **style declaration** sets the **text color** to **blue**.
When to use the methods described above:

- Use Method 1, **External Style Sheets**, to apply a common set of design, layout, and display styles globally to all site pages. This is the most efficient way to implement CSS.
- Use Method 2, **Internal Style Sheets**, when there are a few CSS rules that will only be applied to elements within the current HTML document.
- Use Method 3, **Inline Styles**, when a CSS rule will only be applied to a single element in the current HTML document.

*Note:* CSS rules declared using Methods 2 and 3 can override rules that have been applied using External Style Sheets. This can be useful in situations where a *global* CSS rule does not result in the desired styling for a *particular* element in the current HTML. The next section – **Inheritance, Cascade, and Specificity** – explains how CSS rules are applied, including circumstances which will override previously declared rules.

**INHERITANCE, SPECIFICITY, and CASCADE:**
CSS is applied to HTML elements following rules of Inheritance, Specificity, and Cascade.

**INHERITANCE:**
Some CSS rules are inherited down the *HTML Document Tree* (see above), while others are not. CSS inheritance, when understood, can be used to reduce the number of CSS rules required.

The HTML `<body>` element is the ancestor of all other elements, which means that any inheritable CSS styling properties that are assigned to the `<body>` element will be inherited throughout the rest of the HTML document tree.

*Example:* `body {font: bold; color: red;}`
In this example the *HTML element* `body` has the `font property` set to a value of `bold` and the `color property` set to `red`. Unless overridden, a browser will display all text `bold` and `red`.

Reference material listing CSS properties that are, and are not, inheritable - organized by categories: [http://www.javascriptkit.com/dhtmltutors/cssreference.shtml](http://www.javascriptkit.com/dhtmltutors/cssreference.shtml)

**SPECIFICITY:**
CSS rules may, and often do, conflict with one another. One of the ways these conflicts are resolved is through CSS specificity, which assigns importance (aka specificity) values to the CSS selector types, as follows - listed in order of importance:

1. **ID selectors**
2. **Class** and **Pseudo Class** selectors
3. **Element selectors**

When CSS rules conflict with one another, the most important (or specific) selector is the one that will be applied.

**Calculating Specificity Level**

There is a relatively simple technique to recognize the specificity level of CSS rules that have been assigned to elements in an HTML document.

*The Specificity examples below are based on the following simple CSS rule declarations:*

- div {color: red;}
  - *Element selector* name = div, property = color, value = red

- .link {color: green;}
  - *Class selector* name = link, property = color, value = green

- #nav {color: blue;}
  - *ID selector* name = nav, property = color, value = blue

The examples:

- **Recap**: An ID selector will override an element selector or a class selector, and a class selector will override an element selector.

  **Example 1**: `<div> Some text. </div>`
  The element selector div is set to display red text, and there is no class selector or ID selector, so the text will be rendered in red.

  **Example 2**: `<div class="link"> Some text. </div>`
  The element selector div is set to display red text, the class selector link is set to display green text, and there is no ID selector. The class selector takes precedence (highest specificity), so the text will be rendered in green.

  **Example 3**: `<div class="link" id="nav"> Some text. </div>`
  The element selector div is set to display red text, the class selector link is set to display green text, the ID selector nav is set to display blue text. All three selector types are present, but the ID selector takes precedence (highest specificity), so the text will be rendered in blue.

  **Example 4**: `<div class="link"> 
    <p id="nav"> Some text. </p>
  </div>`
  The element selector div is set to display red text, the class selector link is set to display...
Example 5: `<div class="link">
  <p> Some text. </p>
</div>`

The `element selector div` is set to display red text, the `class selector link` is set to display green text, and there is no ID selector. The `<p>` element, as a child of the div element, inherits properties from its parent (div). The div element has been assigned the class selector link, which takes precedence (highest specificity), so the text will be rendered in green.

Example 6: `<div class="link">
  <p> Some text. </p>
  <p id="nav"> More text. </p>
</div>`

The `element selector div` is set to display red text, the `class selector link` is set to display green text, the ID selector nav is set to display blue text. Here, the div element has been assigned the class named link, with the property value green. The first `<p>` element is a child of the div element, so it inherits the parent properties and that text will be rendered in green. The second `<p>` element has been assigned the ID selector nav, which takes precedence (highest specificity) and overrides the inherited color property, so that text will be rendered in blue.

CASCADE:
CSS rules cascade by order of importance, based on the method used to define the rules. When there are CSS rule conflicts, the rule from the style sheet with highest importance is applied, as follows - listed by order of importance:

1. `!important` - If a rule declaration, no matter where it resides, includes !important it is assigned the highest importance and overrides any conflicting rules.
2. `Inline styles` - Author declared styles located within a specific HTML element tag.
3. `Internal style sheets` - Author declared styles located within the HTML head section.
4. `External style sheets` - Author declared styles located in separate linked CSS documents. 
   Note: If more than one external style sheet is linked then the last one referenced takes precedence.
5. `User style sheet` - Styles declared using an assistive tool that allows users to customize the display of web content by writing their own CSS styles.
6. Default browser style sheet – The default styles built into a browser.

Browsers have a built-in default style sheet, which is designated as the least important. When a browser reads an HTML document that includes author declared styles (Inline, Internal, or External) those rules override the default browser style sheet rules. This is cascade in action.

There are circumstances that result in two or more CSS rule declarations being applied to the same element, and it is possible that this may result in a conflict. When conflicts like this occur, the declaration with the most weight is used. So, how is weight determined?

How a browser applies the Cascade Rules to HTML:

1. Read all style sheets (browser default, user declared, and author declared), and find all declarations whose selectors match a particular element in the HTML.
2. Sort these declarations by weight and origin. For normal declarations, the author style sheets will override both the user style sheets and the browser style sheet (unless the !important declaration in the user's style sheet, which will override author and browser style sheets).
3. Sort the selectors by specificity. More specific selectors will override more general ones.
4. Sort by order of occurrence. If conflicting rules have the same weight, origin, and specificity, then the one encountered last is applied.

As rule 4 indicates, the order in which CSS rules are declared matters. If conflicting rules reside in the same style sheet, each with the same level of specificity, the rule that is declared last in the CSS document will be the one that is applied.

COMMON CSS DECLARATIONS:
The following CSS properties are used to style nearly every HTML document:

Display
Every HTML element can be thought of as residing in a “box”, and the properties of that “box” (size, padding, margins, color, etc.) will determine how that element is displayed in a browser. The CSS display property specifies how this “box” relates to other elements on the web page, as well as the behavior of its child elements (additional elements nested inside of it).

Example: body {display: flex;}
In this example the targeted selector is the HTML element body and the declaration specifies that the display option to be applied is flex.
Common display options: (many more can be found on the W3 Schools website)

- **block;**
  Block level elements, by default, take up as much space as they can and they cannot be placed on the same horizontal line with other elements having any other display mode, including other block elements (unless they are floated). Block elements allow you to alter their width and height properties.

- **inline;**
  Inline level elements can be placed side by side on the same line as other inline elements, so they are displayed in a browser as a single line (wraps when the right edge of the browser window is encountered). Inline is the default mode for all elements that are not specified with any other display values. You will not be able to change an inline element’s width and height properties.

- **inline-block;**
  The Inline-block display value combines the properties of both block elements and inline elements. You get the ability to set a height and width properties, and the elements can appear on the same horizontal line as other elements.

- **flex;**
  The Flex display value will align child elements one next to each other in a horizontal row, or vertical column. The flex-direction - row, or column - can be set using CSS properties.

- **grid;**
  The Grid display value makes the assigned element a grid container, and its child elements behave as grid items.

- **none;**
  The None display value hides the elements so they are not displayed in the browser. This is very useful for Dropdown Menus, where additional options appear when you hover over navigation menus. The elements are set to a display value of none initially, and then the display value is changed to block when the cursor “hovers”.

**Width and Height**

CSS width and height declaration property names are used to set the displayed width and height values of HTML elements, when viewed in a browser. It is possible to set values that are fixed (a specified number of pixels, for example) or values that are relative (percentage, for example).
Web pages are commonly viewed using many different devices, each having different viewport dimensions (the display screen size). It is recommended to use relative size values whenever possible. See [https://www.w3.org/Style/Examples/007/units.en.html](https://www.w3.org/Style/Examples/007/units.en.html) for information on which units of measurement to use in different situations.

**Common units of measurement for Width and Height:**

- **px** – Pixels.
- **pt** – A unit of measurement for the size of printed characters, 12pt for example.
- **em** – A unit of measurement, where 1 em = the *current* font size.
- **rem** – A unit of measurement, where 1 rem = the *root* font size.
- **%** – A percentage of the parent container element size.
- **vh** – A unit of measurement, where 100 vh = 100% of the height of the device viewport.
- **vw** – A unit of measurement, where 100 vw = 100% of the width of the device viewport.
- **auto** – Automatically calculates the value, based on current conditions.

*Example:* `body {width: 90vw;}`

Here the **HTML element selector** is `body` and the **declaration** sets the **property name** `vw` (viewport width) **value** to 90%. In other words, all elements in the body of the HTML document will display using no more than 90% of the viewport width, assigning blank space on both sides of the displayed content.

See [https://www.w3schools.com/cssref/css_units.php](https://www.w3schools.com/cssref/css_units.php) for additional possible units of measurement to use, organized by fixed and relative categories.

Properties like **max-width**, **min-width**, **max-height**, and **min-height** are useful when developing **responsive** websites, where the display of HTML elements adjusts in size and/or arrangement based on the device viewport – expanding on desktops and laptops, shrinking on tablets and phones.

Setting the width of an element to a *relative* value, and the height of that element to *auto*, will adjust the displayed size of that element to fit the container width (the parent element width), and automatically adjust the height of the element, based on the device viewport.

*Example:* `img {max-width: 50%; height: auto;}`

In this example the **HTML element selector** is `img` (image) and the **declaration** sets the **property max-width value** to 50%, and then sets the **property height value** to *auto*. In other words, the image element will display using 50% of the container element width, and the height of the image will adjust automatically. The **actual size** of the image file – the actual pixel width and height – does not change, only the **displayed** size is adjusted.
Margins and Padding

The CSS properties named margin and padding are used to set spaces between HTML elements, using the same value units as the width and height properties. The margin property sets the amount of space to be displayed outside of the target element, while the padding property sets the amount of space to display inside the target element. The following simple HTML code will be used to illustrate how CSS margin and padding properties influence element spacing.

```html
<div>
    <img src="image.jpg">
</div>
```

The HTML above has a division element (div) that contains an image element (img). This will display in a browser simply as an image. Using CSS, you can add a border around the image:

**CSS example 1:**
```css
img {border-width: 1px; border-style: solid; border-color: black;}
```

In this example the CSS element selector is img (image) and the declaration sets the border-width property to 2px (2 pixel wide) and then sets the border-style property to solid (a solid line) and then sets the border-color property to black. This will display as the same image, but now with a 2-pixel wide, solid line, black border added along the outside edges of the image.

Again using CSS, you can use the padding property to add space between the image and the border, and use the margin property to add space outside the border:

**CSS example 2:**
```css
img {border-width: 1px; border-style: solid; border-color: black; padding: 25px; margin: 10px;}
```

Here the padding property, set to 25px (25 pixels), and the margin property, set to 10px (10 pixels), have been added. This results in 25 pixels (padding) between the image and the border, and 10 pixels (margin) between the border and the outside edge of the parent <div> element, which expands to accommodate the added space.
In addition to adding space between elements, the margin property can be used to horizontally center elements on the page:

**Example:** `.centered {margin: auto;}

This example declares a class named centered with the margin property set to auto. Any HTML element that is assigned the centered class will be horizontally centered in the browser viewport (unless another CSS rule overrides).

*There are far too many CSS properties, rules, and uses to attempt to present them all in this document. However, on-line resources are readily available. Here are a few:*

https://www.w3.org/Style/CSS/#specs


http://web.simmons.edu/~grabiner/comm244/weekfour/css-concepts.html


https://blog.hubspot.com/website/what-is-css-class

https://www.w3schools.com/css/css_intro.asp


https://www.sfu.ca/~tutor/techbytes/Dreamweaver/dw2.html

https://blog.hubspot.com/website/css-display-property