Module 3
Week 1 Review

HTML

HTML (Hypertext Markup Language) is a markup language that is used to describe how a document should be rendered by a web browser.

- Markup is dome using **element tags.**
  - Opening Tag
  - Closing Tag (if not a self closing tag)

HTML Document Structure

- `<!DOCTYPE html>`
- `<html>`
- `<head>`
- `<body>`

Semantic Elements and Organization

- There are elements that exist to group similar content together. These are called **semantic elements.**

- This is different from using an **`<h1>`** to define the heading—though that's still valid—as these elements have no visual display or indicator. Of course, you can always add styles if you wish.

Semantic Element Examples

- `<header>`
- `<main>`
- `<section>`
- `<article>`
- `<nav>`
- `<footer>`

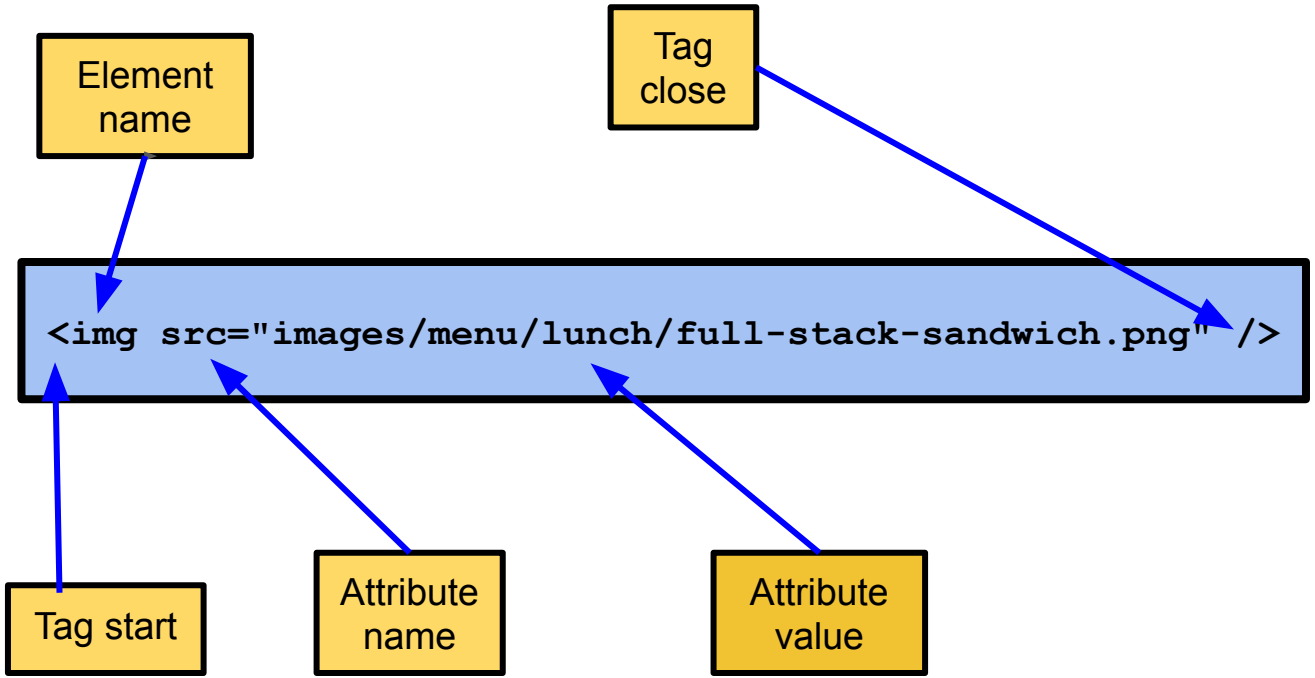
Attributes

- **Attributes** are pieces of code that are included within the tag definition (within the tag's angle brackets).

- Often used to "extend" a tag by changing its behavior or providing metadata.

- Usually in the format `name = value` but there are a few instances where you can use shorthand and omit the value (it's a good idea to include it anyway though).

The <img> Tag

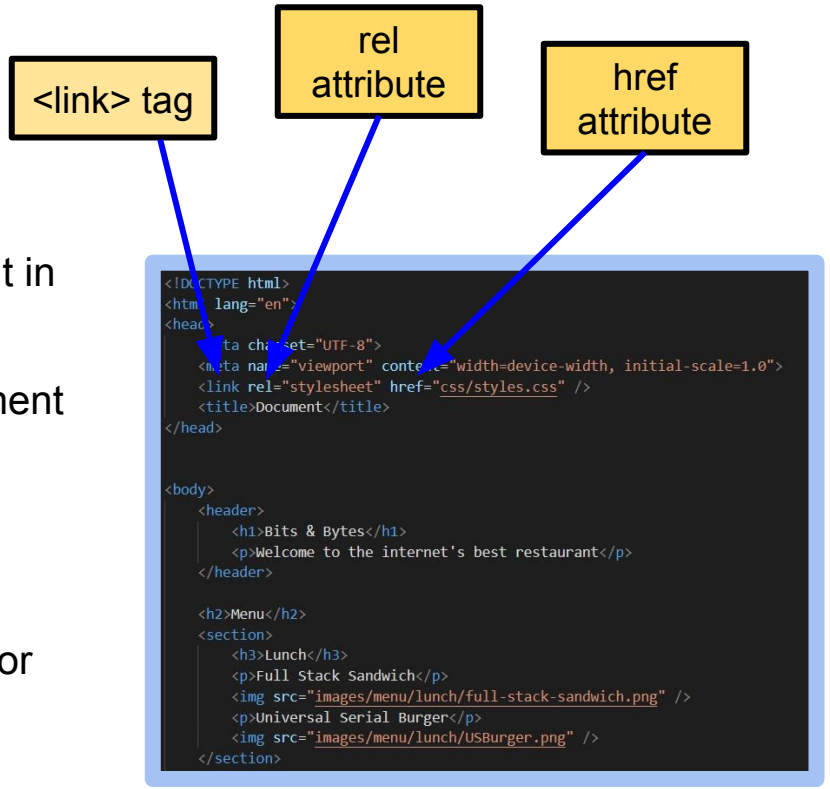


- The **<img>** element is used to display an image on an HTML page.
- The **src** attribute is used to specify the "source" image by providing the relative file path to the image file.
- The **<img>** element is used to specify a resource rather than contain child data, so it is "self-closing" - we close the single tag with a space and **/>**

CSS

- We use **Cascading Style Sheets (CSS)** to style HTML.
- CSS contains only styling information.
- We tell CSS which elements we want to affect using **selectors**. Anything that matches the selector of a CSS rule will be affected by it.

Adding CSS to an HTML Document



In order for a CSS document to be used by an HTML document, we need to add a **link** to the CSS document in the HTML.

To do this, we place a **`<link>`** tag in our **`<head>`** element (**where** in the **`<head>`** element doesn't matter).

`<link>` has two attributes:

- **`rel`**
  - Stands for "**rel**ationship" to the HTML file. For CSS files, we use the value "**stylesheet**"
- **`href`**
  - Stands for "**h**yperlink **ref**erence" to the HTML file. This is where we put the path to the CSS file.

Font Styling

We can change font styling using the **`font-family`** property.

```
body {
    font-family: Arial, Helvetica, sans-serif;
}
```

- It's typically a good idea to use **web safe fonts**.
- Can specify fonts that can be downloaded from web font providers.
- Example above shows **font name** as well as some **fallback options**.
- **Always include a fallback web safe font.**

Font Styling

- **`font-size`**
- **`color`**
  - Color can be specified in several ways
    - **keyword** (currently about 140 options such as **`blue`**, **`darkred`**, etc)
    - Six digit **hexadecimal** value (i.e. **`#0000FF`**, **`#8B0000`**, etc.)
    - Read, green, blue (**rgb**) value (i.e. **`rgb(0, 0, 255)`**, **`rgb(139, 0, 0)`**, etc)

Forms

Forms are one way to submit data from the client to the server.

You've likely encountered forms on the web when logging in to websites, posting on social media, or completing an e-commerce purchase.

Forms are enclosed within a **`<form>`** element.

- **`<form>`** element has at least two attributes:
  - **`method`**
    - **`GET`**
    - **`POST`**
  - **`action`**
    - URL the form will send its data to (an API endpoint for example)

Notes on Using GET as method

More about using **`GET`** as the **`method:`**

- Form data is appended into the URL in name/value pairs.
- **Never use `GET` to send sensitive data**. It will be visible in the URL.
- Useful for form submissions where a user wants to bookmark the result.
- **`GET`** is better for non-secure data, like query strings for a search engine.

Notes on Using POST as method

More about using **`POST`** as the **`method:`**

- Form data is sent inside the body of the HTTP request (data is not shown in URL).
- Form submissions with **`POST`** cannot be bookmarked.
- Just like **`POST`** methods on the server, it should be reserved for "creating" things like reservations.

Input Elements

```
<input type="text" />
```

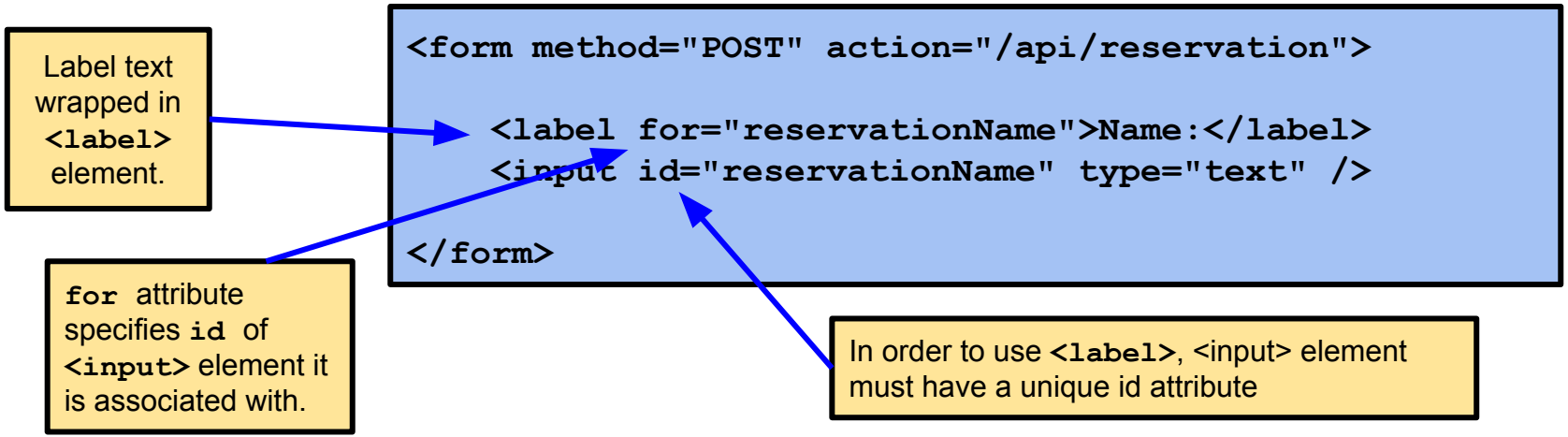
Some examples of the type of <input> elements we can use:

- **`text`**
- **`number`**
  - Forces the input data to be numeric
- **`date`**
  - Provides a date picker
- **`time`**
  - Provides a time picker
- **`checkbox`**
  - allows you to check if box is checked or not

Form labels

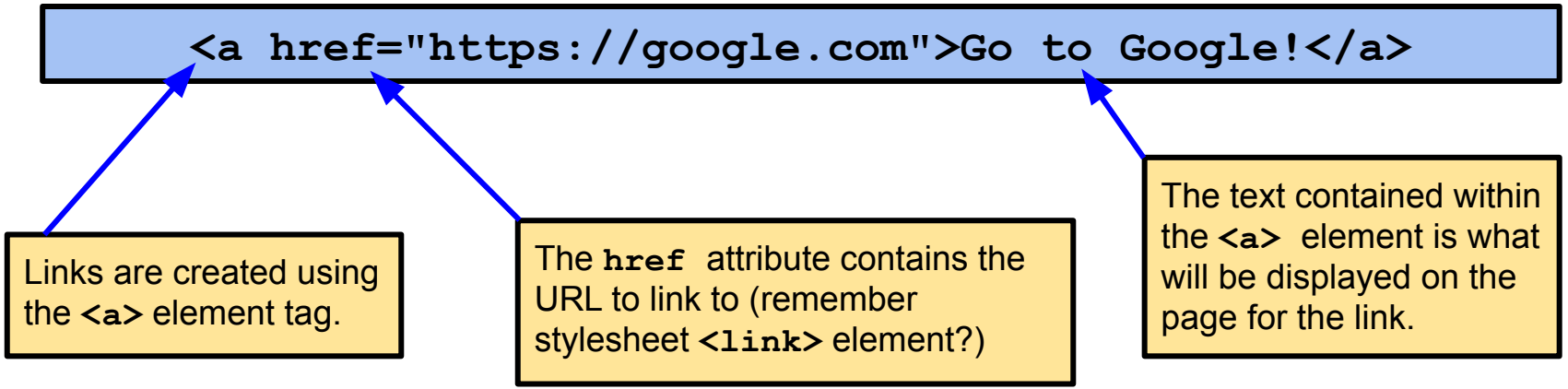
Form elements typically require a label of some sort to indicate what the input field is asking for.

HTML provides a **<label>** element which is associated with an input elements.



Using Nav and Hyperlinks

- You can use the **<nav>** element to give semantic meaning to the navigation portion of an HTML page.
- Navigation is often accomplished by using hyperlinks. Hyperlinks are all over the web and they are easy to add to your code.



Form Element Names

Form elements should also have a **`name`** attribute.

```
<input id="firstName" name="firstName" type="text" />
```

The element **`id`** is often used by CSS/Javascript but when a form is submitted to backend code as a set of form fields, the fields will be identified by their **`name`** attribute.

HTML Hierarchy

- Parent
- Child
- Siblings
- Descendants

Styling with CSS

- Inline **`<style>`** element
- **`<style>`** element in document **`<head>`**
- External doc(s) using
  - **`<link rel="stylesheet" href="" />`**

CSS Rule

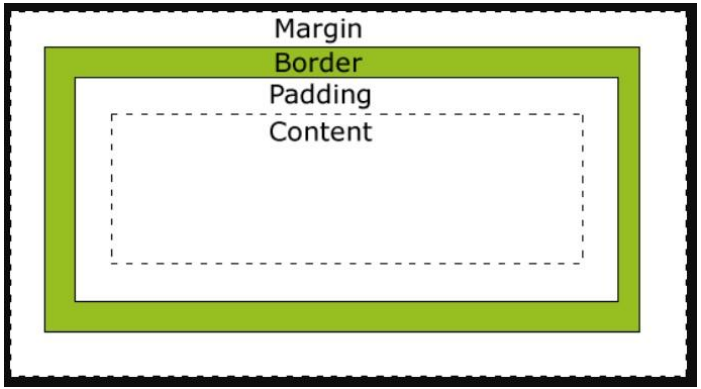
- Selector
  - * (Universal)
  - Element
  - # ID
  - . Class
- Declaration block

Advanced Selector Types

- **Combinations**
  - `div.warning` (`div` elements with `warning` class)
- **Multiple Selector (comma)**
  - `h1, h2` (`h1` elements and `h2` elements)
- **Descendant Selector (space)**
  - `body h1` (`h1` descendants of `body`)
- **Child selector (>)**
  - `body > header` (`header` children of `body`)
- **Attribute Selector**
  - `input[type=number]` (`input` elements whose `type` attribute has a value of `number`)
- **Pseudo-class**
  - `a:hover` (`a` elements in `hover` state)

Box Model

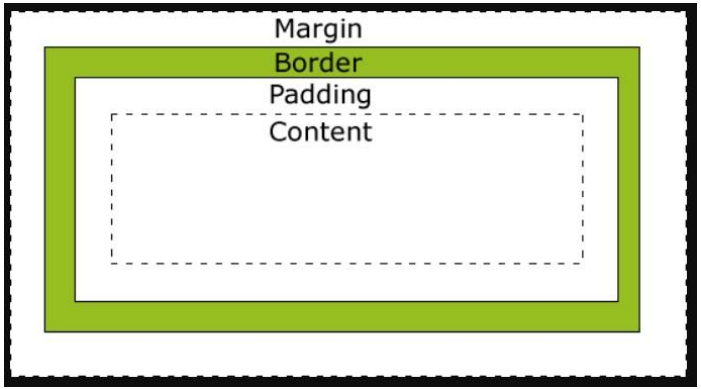
- Every **element** in web design **is** a **rectangular box**.
- The **content, padding, border, and margin** can be used to calculate the amount of space that an element takes up.
- Margin is the space **outside an element**. It does not affect the size of the box but affects other content that interacts with the box.
- Padding is the space **inside an element**.



Box Model

The **`box-sizing`** css property allows you to change how the width and height of an element are calculated. The possible values for this property are:

- **`content-box`** (default) includes only size of element in size calculations.
- **`border-box`** includes content, padding, and border of element but do not include the margin, when calculating size.



Display Property

- `block`
- `inline`
- `inline-block`
- `none`

Positioning

- Normal flow (**`static`**)
  - Left to right, top t bottom
- **`relative`**
- **`absolute`**
- **`fixed`**

Units of Measurement

- Pixels
    - i.e. **`40px;`**
- Other absolute units:
    - i.e. **`in, cm, mm, pt, pc`**
- Relative to the current font
    - i.e. **`2.5em;`**
- Relative to the Root Element font
    - i.e.**`1.8rem;`**
- Percentage relative to parent
    - i.e. **`width: 50%`**

Aside: CSS Variables

You can set up variables that can be used in your CSS to make changing values easier.

```
:root {
    --main-bg-color: gray;
}

body {
    background-color: var(--main-bg-color);
}
```

Grid Containers

- **display: grid;**
- **grid-template-columns**

```
grid-template-columns: 20px 100px 50px;
grid-template-columns: 100px 1fr 1fr;
```

- **grid-template-areas**

```
grid-template-areas:
    "name1 name2 name3"
    ". name4 name 5"
    "name6 name6 name6";
```

Grid Containers

- **`grid-gap: 10px;`**
- **`grid-area`**

```
header {
    grid-area: header;
}
```

- Used in **`grid-template-areas`**

```
grid-template-areas:
    "header header header"
    ". name4 name 5"
    "name6 name6 name6";
```

Grid Containers

- **`grid-template-rows`**

```
grid-template-rows: 100px 1fr 100px;
```

- Grid items are **direct children** of the grid container in the HTML.

Grid Alignment

- **`justify-items`**
  - aligns items along the row axis
    - **`start`**
    - **`end`**
    - **`center`**
    - **`stretch`**

Grid Alignment

- **`align-items`**
  - aligns items along the column axis
    - **`start`**
    - **`end`**
    - **`center`**
    - **`stretch`**

Grid Alignment

- **`justify-self`**
  - overrides row alignment for element.
- **`align-self`**
  - overrides column alignment for element.

Responsive Design

A page should respond to the user's environment based on screen size, platform, and orientation (landscape / portrait).

Consider:

- Flexible grid layout
  - Use percentages, not pixels
- Responsive images
  - Use percentages, not pixels

```
img {
     width: 100%;
     height: auto;
}
```

- Change layout entirely based on screen characteristics

Media Queries

Selectively apply CSS based on screen/environment characteristics.

- **`@media`** rule
  - https://www.w3schools.com/cssref/css3_pr_mediaquery.asp
- Apply CSS conditionally based on **`@media`** query

```
@media screen and (max-width: 768px) {
    .container {
        // override properties
    }
};
```

Breakpoints

- The width where something changes
- Common examples:
  - Small devices (phones - **`min-width: 576px`**)
  - Medium devices (tablets - **`min-width: 768px`**)
  - Large devices (desktop - **`min-width: 992px`**)
  - Extra large devices (large desktops - **`min-width: 1200px`**)

Flexbox

- **`display: flex`**
- Direct children of container are flexbox items
- **`flex-direction`** is **`row`** (default) or **`column`**
- **`align-items`** - vertical alignment
- **`justify-content`** - horizontal alignment
- When **`flex-direction`** is **`column`**
  - **`align-items`** - horizontal alignment
  - **`justify-content`** - vertical alignment

Values for justify-content

- `flex-start`
- `flex-end`
- `center`
- `space-between`
- `space-around`
- `space-evenly`

Values for align-items

- `stretch`
- `flex-start`
- `flex-end`
- `center`

ORDER AND ROW-REVERSE

- **`flex-grow`**
  - Allows element to grow
  - 0 = false
  - 1 …. x = grow ratio
- **`flex-shrink`**
  - Allows element to shrink
  - 0 = false
  - 1 …. x = shrink ratio
- **`flex-basis`**
  - Defines the default size of an element before the remaining space is distributed.

FLEX-GROW, FLEX-SHRINK, FLEX-BASIS

- **`order`**
  - Can specify **flex-item order** with numbers in this property

- **`Row-reverse`**
  - can be specified as **`flex-direction`**

FLEX-WRAP

- `nowrap`
- `wrap`
- `wrap-reverse`