HTML Class:

HTML (HyperText Markup Language) is the standard language used to create and structure content on the web.

HTML was first introduced by Tim Berners-Lee in 1991. He is also the creator of the World Wide Web.

HTML defines the structure of a webpage. It organizes content into different sections (like headings, paragraphs, tables, and lists), which helps browsers render the page properly

HTML is used purely for structure, whereas CSS (Cascading Style Sheets) is used for design (styling) and JavaScript is used for interactivity (functionality).

HTML DOcument:

All HTML documents must start with a document type declaration: <!DOCTYPE html>.

The HTML document itself begins with <html> and ends with </html>.

The visible part of the HTML document is between <body> and </body>.

The <meta> tag is essential for providing metadata about a webpage.

Proper use of <meta> tags helps ensure better functionality, performance, and discoverability of your web pages.

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
```

HI

</body>

</html>

HTML Elements and Tags:

- **Block-level Elements**: , <header>, <footer>, <article>, <section>, etc.
- **Inline Elements**: , <a>, , , etc.
- **Forms**: <form>, <input>, <textarea>, <button>, <select>, <option>, <label>, etc. React makes form handling dynamic, so understanding these elements is crucial.
- **Links and Navigation**: <a>, <nav>, and how React Router handles navigation in SPAs
- The HTML <**p**> element defines a paragraph.
- The <**hr**> element is used to separate content (or define a change) in an HTML page
- The HTML <**br**> element defines a line break.
- The text inside a <**pre**> element is displayed in a fixed-width font (usually Courier), and it preserves both spaces and line breaks

HTML Headings:

<h1>Heading 1</h1>
<h2>Heading 2</h2>
<h3>Heading 3</h3>
<h4>Heading 4</h4>
<h5>Heading 5</h5>
<h6>Heading 6</h6>

> Setting the style of an HTML element can be done with the style attribute.
>
> <body style="background-color:powderblue;">

HTML Formatting Elements tags:

- <b> - Bold text
- - Important text
- <i> - Italic text
- - Emphasized text
- <mark> - Marked text

- <small> - Smaller text
- <del> - Deleted text
- <ins> - Inserted text
- <sub> - Subscript text
- <sup> - Superscript text

Attributes and Properties:

- **HTML Attributes**: id, class, src, href, alt, etc.
- **Class vs. className**: In JSX, class is replaced with className because class is a reserved word in JavaScript.
- **Style Attributes**: How style is handled in React, using an object with camelCase property names instead of the string format used in HTML.

Semantic HTML Tags: Semantic HTML elements are tags that clearly describe their meaning both to the browser and to the developer

- **Heading Tags**: <h1>, <h2>, etc.
- **Lists**: <ul>, <ol>, <li>, and when to use them.
- **Forms**: Proper use of <label>, <fieldset>, <legend>, etc.
- **Accessibility**: Adding ARIA roles and attributes to improve accessibility, which will be important in React as well.

Non-Semantic HTML: Non-semantic HTML elements are tags that do not describe the content they enclose. These elements don't provide any meaning about the content and are used primarily for layout, styling, or grouping purposes.

 <i> <u>

HTML Event Handling

- **HTML Event Attributes**: onclick, onchange, onsubmit, etc.
- **React Event Handling**: onClick, onChange, etc.

HTML Quotation and Citation Elements:

- <blockquote>: element defines a section that is quoted from another source
- <q>: The HTML <q> tag defines a short quotation.
- <abbr>: tag defines an abbreviation or an acronym, like "HTML", "CSS",
- <address>: tag defines the contact information for the author/owner of a document or an article
- <cite>: tag defines the title of a creative work (e.g. a book, a poem, a song, a movie, a painting, a sculpture, etc.).
- <bdo>: The HTML <bdo> tag is used to override the current text direction

HTML Comment Tag:

<!-- Write your comments here -->

HTML Colors:

RGB(*red, green*, *blue*)
hexadecimal: *#rrggbb*
HSL(*hue*, *saturation*, *lightness*)

HTML Styles - CSS:

- **Inline** - by using the style attribute inside HTML elements
- **Internal** - by using a <style> element in the <head> section
- **External** - by using a <link> element to link to an external CSS file

Forms and Input Handling:

- **Input Elements**: <input>, <textarea>, <select>, and the **"controlled" component pattern** in React (where React state is used to control input elements).
- **Form Submissions**: Handling form submissions in React, including event prevention (event.preventDefault()).

HTML Iframe Syntax:

The HTML <iframe> tag specifies an inline frame.

An inline frame is used to embed another document within the current HTML document.

HTML File Paths:

A file path describes the location of a file in a website's folder structure.

File paths are used when linking to external files, like:

- Web pages
- Images
- Style sheets
- JavaScripts

1. **Absolute Path**: An **absolute path** refers to a full path that specifies the location of a file or resource starting from the root directory. It provides the complete address to access the file or resource.
   EX: /Users/john/Documents/Projects/myWebsite/index.html

2. Relative Path: A relative path is a path that describes the location of a file or resource in relation to the current directory. It does not start from the root directory and instead depends on where the current file is located.

EX: myWebsite/index.html

Differences between HTML and HTML5

Feature	HTML (HTML4)	HTML5
Doctype Declaration	<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">	<!DOCTYPE html>
Audio and Video Support	Not natively supported (requires plugins like Flash).	Native support with <audio> and <video> tags.
Canvas	Not available.	<canvas> tag for drawing graphics via JavaScript.
Form Elements	Limited input types (text, password, etc.).	New input types (e.g., email, tel, date, url, range, etc.) for better form validation.
Semantic Tags	No semantic elements (e.g., <header>, <footer>)	New semantic elements like <header>, <footer>, <article>, <section>, <nav>, <aside>, etc.
Local Storage	Not supported.	localStorage and sessionStorage for client-side storage.
Geolocation	Not supported.	navigator.geolocation API for geolocation services.
Offline Web Applications	Not supported.	The ability to create offline web apps with Application Cache and later with Service Workers.
Web Workers	Not supported.	Multi-threading support through Web Workers.

SVG Support	Can be used via <object> or <embed> tags.	Full native support for SVG (Scalable Vector Graphics).
Web Fonts	No native support.	Native support for web fonts via the <link> and @font-face rules in CSS.
APIs	Limited APIs.	New APIs such as Canvas API, Audio API, Geolocation API, WebSockets API, Web Storage API, etc.
Inline SVG	Not supported.	Inline SVG is fully supported.
Doctype	More complex with stricter syntax and document type definitions.	Simplified <!DOCTYPE html>.
Character Encoding	Often required meta tag for specifying character encoding.	Default UTF-8 character encoding (no need for explicit declaration).
CSS3 Support	Limited support (CSS2.1).	Full support for CSS3 features (animations, transitions, media queries, etc.).
Data Attributes	Not supported.	Supports custom data-* attributes for storing extra information on HTML elements.
APIs for Mobile and Device Features	Not available.	Access to device APIs for mobile apps (e.g., accelerometer, camera).
Video Formats	Flash or third-party plugins were required for videos.	Native support for video formats like .mp4, .webm,

CSS Class

What is CSS: (**Cascading Style Sheets**) is a styling language used to control the appearance and layout of web pages. It works alongside HTML and JavaScript to define how elements are displayed.

Key Features of CSS

- **Selectors:** Target HTML elements (.class, #id, element)
- **Properties:** Define styles like color, font-size, margin
- **Responsiveness:** Media queries for different screen sizes
- **Animations & Transitions:** Smooth UI interactions
- **Frameworks & Libraries:** Bootstrap, Tailwind CSS, Styled Components

Essential CSS Concepts for React Developers

Selectors: Selectors help you target specific elements.

Selector	Example	Explanation
Class	.button { color: red; }	Targets all elements with class button
ID	#header { font-size: 20px; }	Targets the element with id="header"
Element	h1 { color: blue; }	Targets all <h1> elements
Universal	* { margin: 0; }	Applies styles to all elements
Grouping	h1, p { font-weight: bold; }	Targets multiple elements
Pseudo-class	button:hover { background: green; }	Styles an element on a specific state (hover)

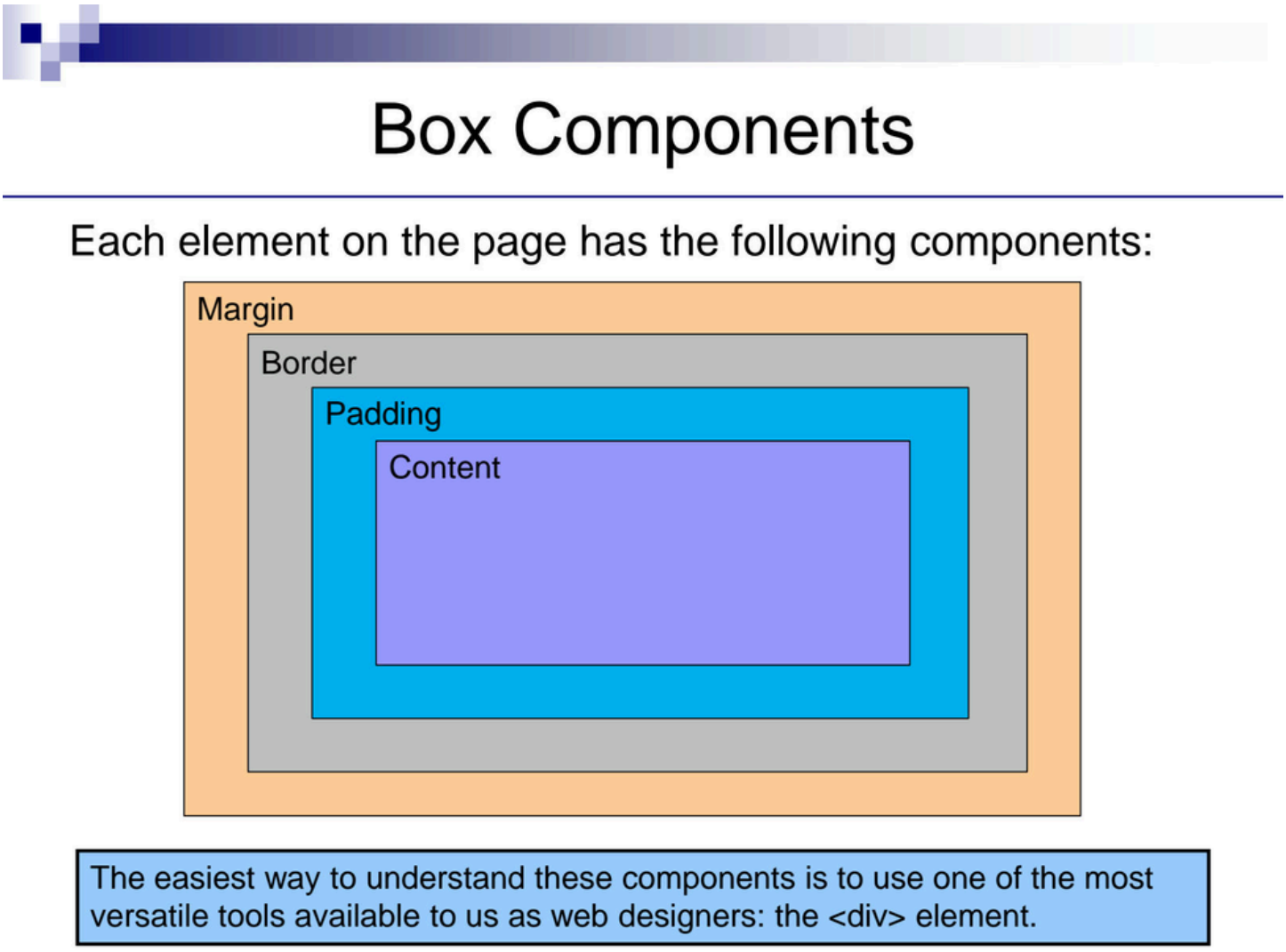
Box Model: The **Box Model** explains how elements are sized and spaced.

Margin - (Outer Space)

Border - (Element Boundary)

Padding - (Inner Space)

Content - (Actual Text/Image)



Display Properties:

- **Block** (div, p) → Takes full width
- **Inline** (span, a) → Takes only needed width
- **Flex** (display: flex;) → For row/column layouts
- **Grid** (display: grid;) → For structured layouts

Responsive Design: Making layouts adaptive to different screen sizes.

```
@media (max-width: 768px) {

 .container {

   flex-direction: column;
```

```
 }

}
```

CSS Units:

Unit	Example	Behavior
px	width: 100px;	Fixed size
%	width: 50%;	Relative to parent
vw	width: 50vw;	50% of viewport width
vh	height: 100vh;	Full screen height
rem	font-size: 2rem;	Based on root size
em	font-size: 1.5em;	Based on parent size

Display Property: Defines how elements appear on the page.

Value	Behavior
block	Takes full width
inline	Takes only necessary width
inline-block	Like inline, but allows height/margin
flex	Enables Flexbox layout
grid	Enables Grid layout
none	Hides the element

Position Property:

Value	Behavior
static	Default positioning
relative	Position relative to itself
absolute	Position relative to nearest positioned ancestor
fixed	Stays fixed on the screen
sticky	Sticks when scrolling

Flexbox:

Property	Description
flex-direction	Row (row), Column (column)
justify-content	Aligns items horizontally
align-items	Aligns items vertically
flex-wrap	Wraps items to next line

CSS Grid:

Property	Description
grid-template-columns	Defines columns (1fr 2fr)
grid-template-rows	Defines rows
gap	Adds space between grid items
place-items	Aligns items

Flexbox (display: flex;) and Grid (display: grid;) are both CSS layout systems used to structure and align elements in a web page. They each have unique use cases and strengths.

Flexbox (display: flex;)

- Designed for **one-dimensional** layouts (either row-based or column-based).
- Works best for aligning items **along a single axis** (horizontal or vertical).
- Allows flexible sizing and spacing between elements.
- Uses properties like justify-content, align-items, and flex-wrap for control.

CSS Grid (display: grid;)

- Designed for **two-dimensional** layouts (both rows and columns).
- Works best when structuring entire page layouts.
- Allows precise control over rows, columns, and gaps.
- Uses properties like grid-template-columns, grid-template-rows, and grid-gap.

Feature	Flexbox	Grid
Layout Type	1D (row or column)	2D (row and column)
Best For	Aligning elements in a row or column	Complex page layouts
Item Control	Items flow naturally and adjust based on content	Defined by explicit row & column structure
Example Use	Navigation bars, buttons, aligning form elements	Page structure, dashboards, grids

When to Use Which?

- **Use Flexbox** when arranging items **in a single row or column**, such as navigation bars or buttons.
- **Use Grid** when designing **structured layouts** with multiple rows and columns, like a full webpage.

Keyframe Animations:

Keyframe animations in CSS allow elements to **gradually change styles** over a defined period using the @keyframes rule. You can define multiple steps within an animation, controlling how an element moves, changes color, fades in/out, and more.

```
@keyframes animation-name {
 from {
  /* Starting Styles */
 }
 to {
  /* Ending Styles */
 }
}
```

CSS Variables: CSS Variables, also called Custom Properties, allow you to define reusable values (like colors, fonts, sizes) that can be updated dynamically across your stylesheet.

```
root {
 --primary-color: blue;
 --text-color: white;
}


.button {
 background: var(--primary-color);
 color: var(--text-color);
 padding: 10px 20px;
 border-radius: 5px;
}
```

Why Use CSS Variables?

✅ **Reusability** – Define once, use multiple times
✅ **Dynamic Styling** – Can change based on themes (e.g., dark mode)
✅ **Easy Maintenance** – Update one variable instead of changing multiple styles

Pseudo Selector:

:active

- Applied when an element is being activated (e.g., clicked).

```
button:active {
  background-color: red;
}
```

: focus

- Applied when an element, such as an input or a link, gains focus (e.g., clicked or tabbed to).

```
input: focus {
  border-color: blue;
}
```

: visited

- Applied to links that have already been visited.

```
a: visited {
  color: purple;
}
```

:first-child

- Targets the first child element of its parent.

```
p:first-child {
  font-weight: bold;
}
```

:last-child

- Targets the last child element of its parent.

```
li:last-child {
  margin-bottom: 0;
```

}

:nth-child(n)

- Selects elements based on their position in the parent. You can use n as a number, or keywords like odd and even.

```
div:nth-child(2) {
 background-color: yellow;
}

ul li:nth-child(odd) {
 background-color: lightgray;
}
```

:nth-of-type(n)

- Selects the nth element of a specific type (ignoring other types of elements).

```
p:nth-of-type(2) {
  color: green;
}
```

:not(selector)

- Selects elements that do **not** match the given selector.

```
div:not(.highlight) {
  background-color: gray;
}
```

:checked

- Applied to checkboxes or radio buttons when they are checked.

```
input:checked {
  background-color: green;
```

}

:disabled

- Applied to form elements that are disabled.

```
input:disabled {
  background-color: lightgray;
}
```

:enabled

- Applied to form elements that are enabled.

```
button: enabled {
  cursor: pointer;
}
```

:empty

- Targets elements that have no children.

```
div:empty {
  display: none;
}
```

:first-of-type

- Selects the first element of a specified type among its siblings.

```
p:first-of-type {
  color: blue;
}
```

:before

- Inserts content before the content of an element.

```
p::before {
  content: "Note: ";
```

```
  font-weight: bold;
}
```

:after

- Inserts content after the content of an element.

```
p::after {
  content: " (end of paragraph)";
}
```

:root

- Targets the root element (<html> in HTML).

```
:root {
  --main-color: blue;
}
```

:lang(language)

- Selects elements based on the language of the document.

```
p:lang(fr) {
  font-style: italic;
}
```

:target

- Applied to an element when it is targeted by a URL fragment identifier (e.g., #section).

```
#section: target {
  background-color: lightyellow;
}
```

px (Pixels)

- **Meaning**:

    - px is an absolute unit that defines the size in pixels. One pixel corresponds to a physical dot on the screen (though the exact size can vary depending on the screen's resolution).
- **Use Cases**:
    - Used when you need precise, fixed-size measurements for elements.
    - Ideal for pixel-perfect designs, where you need exact control over element sizes.

em

- **Meaning**:
    - em is a relative unit that is based on the font size of the **current element** (or its parent if the font size is inherited).
    - 1em is equal to the font size of the element it's used on.

Use Cases:

- Great for scaling elements based on the parent font size. Used primarily for typography and layout adjustments.
- Can be applied to font sizes, padding, margins, and more.

```
.parent {

  font-size: 16px;

}

.child {

  font-size: 2em;  /* This will be 32px (16px * 2) */

}
```

rem (Root em)

- **Meaning**:
    - rem is a relative unit similar to em, but instead of being based on the font size of the element, it is based on the font size of the **root element** (<html>), typically the browser’s default font size (usually 16px).
    - 1rem is always equal to the font size of the root element (<html>), regardless of where it is used in the document.

Use Cases:

- Ideal for defining consistent, scalable sizes throughout the document.
- Typically used for font sizes, spacing, and layout to create responsive designs.

```
html {
  font-size: 16px;  /* 1rem = 16px */
}

.header {
  font-size: 2rem;  /* This will be 32px */
}
```

vh (Viewport Height)

- **Meaning**:
  - vh is a relative unit that represents a percentage of the **viewport height**.
  - 1vh is equal to 1% of the viewport height.
- **Use Cases**:
  - Useful for creating full-height layouts, such as sections that take up the entire height of the browser window.
  - Great for creating responsive elements that adjust based on the viewport height.

```
.full-height {
  height: 100vh;  /* This will make the element take up the entire height of the viewport */
}
```

vw (Viewport Width)

- **Meaning**:
  - vw is a relative unit that represents a percentage of the **viewport width**.
  - 1vw is equal to 1% of the viewport width.
- **Use Cases**:
  - Used for creating responsive layouts that adjust based on the width of the viewport.
  - Commonly used for scaling fonts, widths, or other elements proportionally to the screen size.

```
.responsive-box {
  width: 50vw;  /* This will make the element 50% of the viewport width */
}
```

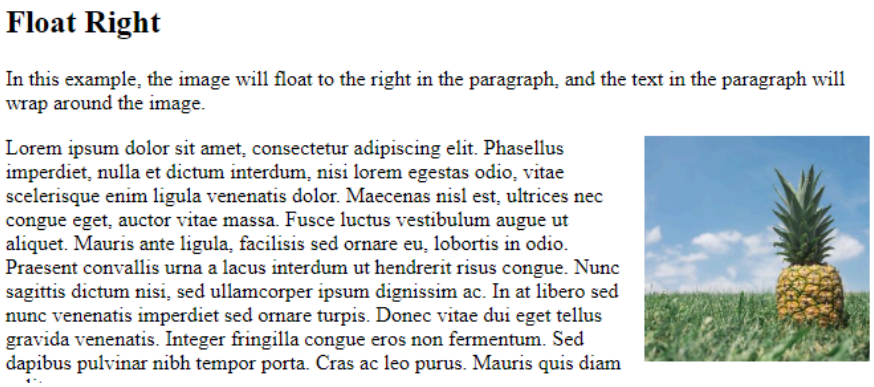
% (Percentage)

- **Meaning**:
  - % is a relative unit that is based on the **parent element's size**. For width, it's based on the parent container's width, and for height, it's based on the parent container's height.
- **Use Cases**:
  - Used for creating flexible layouts, such as fluid grids or elements that need to resize relative to their container.
  - Common in responsive design, where elements adjust their size based on the parent.

```
.container {
  width: 100%;  /* Full width of the parent container */
}
.box {
  width: 50%;  /* 50% of the container's width */
}
```

CSS Layout - float and clear:

The CSS float property specifies how an element should float.



clear:(none,left, right, both etc)

When we use the float property, and we want the next element below (not on right or left), we will have to use the clear property.

The clear property specifies what should happen with the element that is next to a floating element.

CSS Opacity / Transparency:

The opacity property can take a value from 0.0 - 1.0. The lower the value, the more transparent

Conclusion:

- **HTML Basics**: Structure of an HTML document (e.g., doctype, <html>, <head>, <body>, etc.), tags (headings, paragraphs, links, images, etc.), and the importance of semantic tags.
- **CSS Basics**: How to style elements using selectors, properties, and values, including basic styling like colors, fonts, margins, padding, borders, etc.
- **Box Model**: Understanding how margins, borders, padding, and content work together.
- **Layout**: Basic layout concepts such as positioning (static, relative, absolute), flexbox, and grids.
- **Responsive Design**: Introduction to using media queries to make designs responsive.