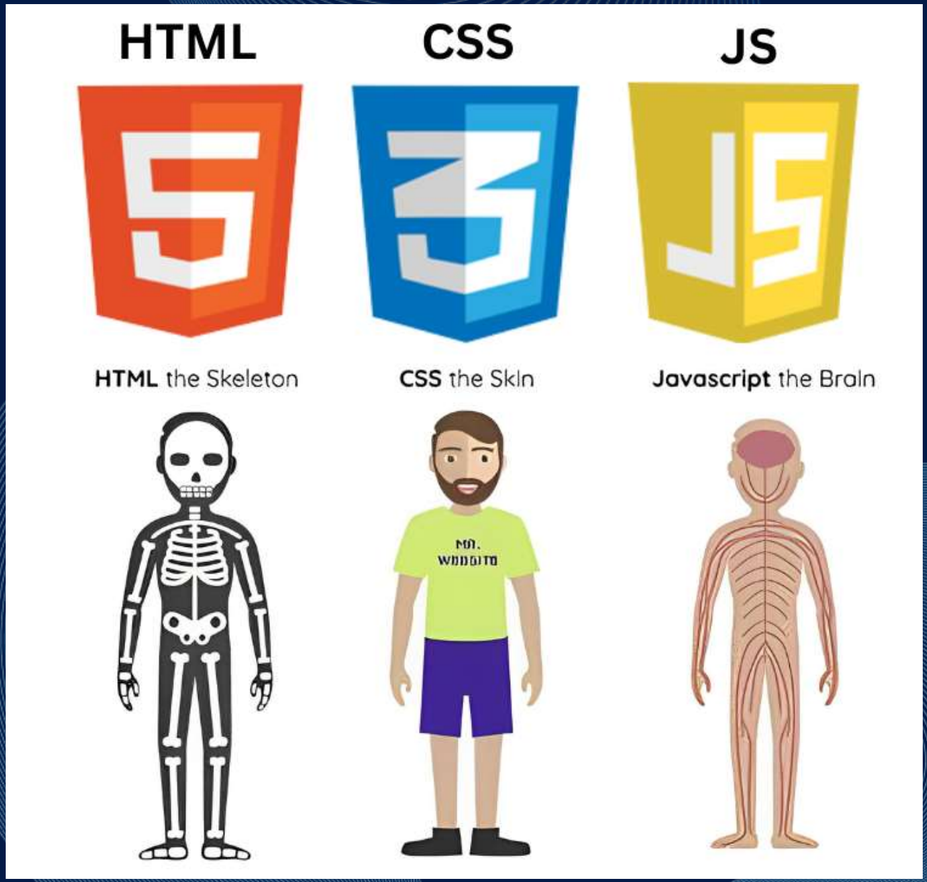
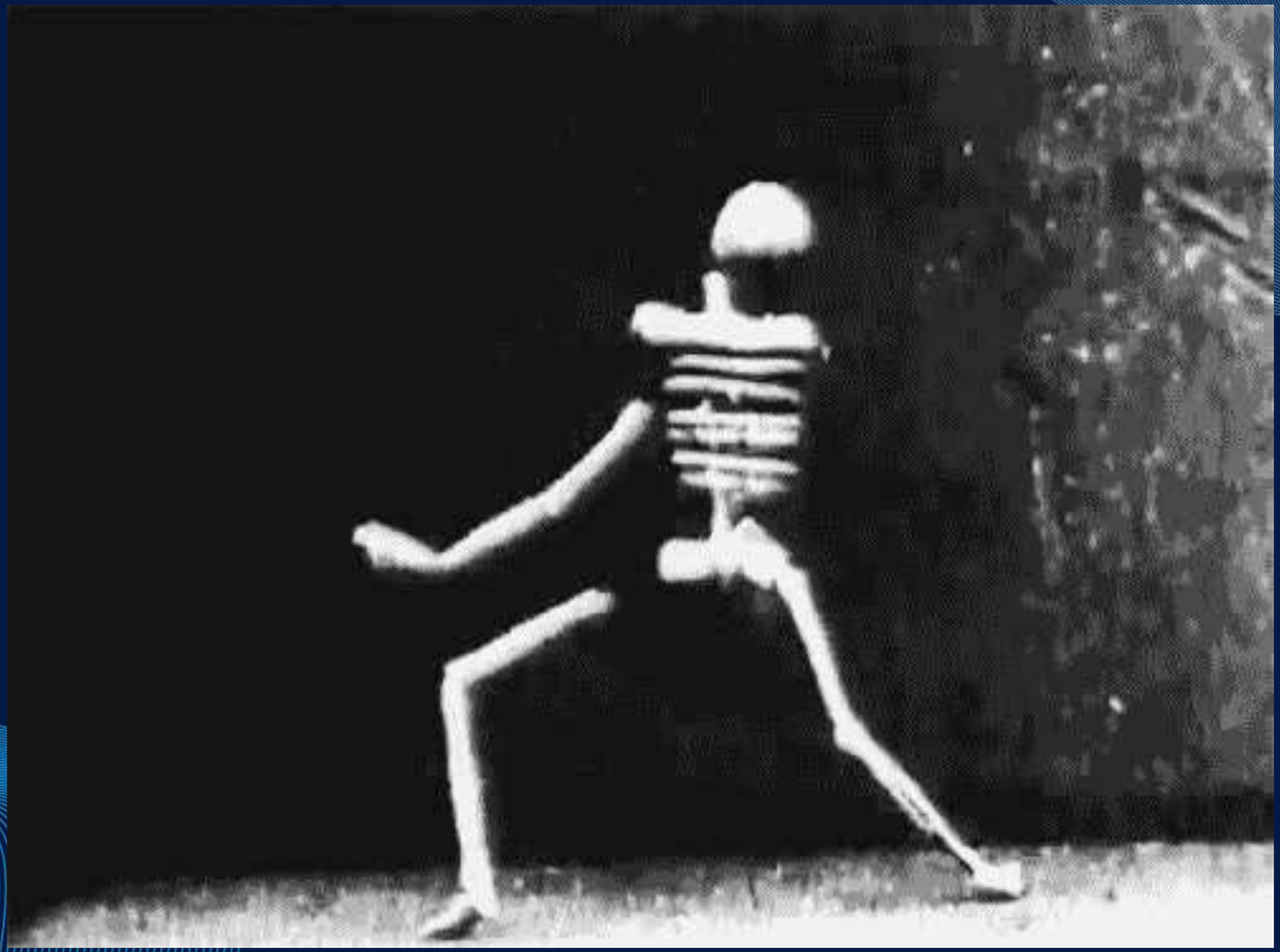
WEB DEVELOPMENT

WHAT IS A WEBSITE MADE UP OF?

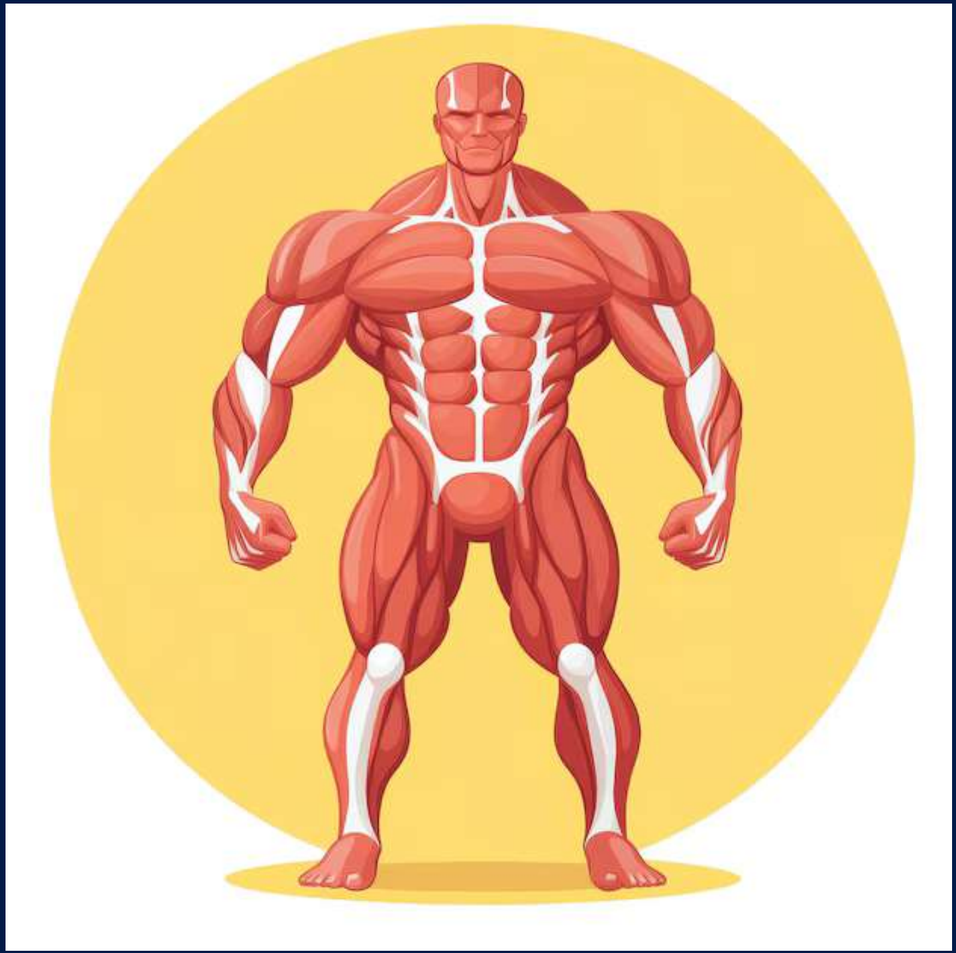
- HTML: The skeleton – provides structure.
- CSS: The skin and style – defines appearance.
- JavaScript: The brain – adds behavior and interactivity.



HTML



CSS



JAVASCRIPT

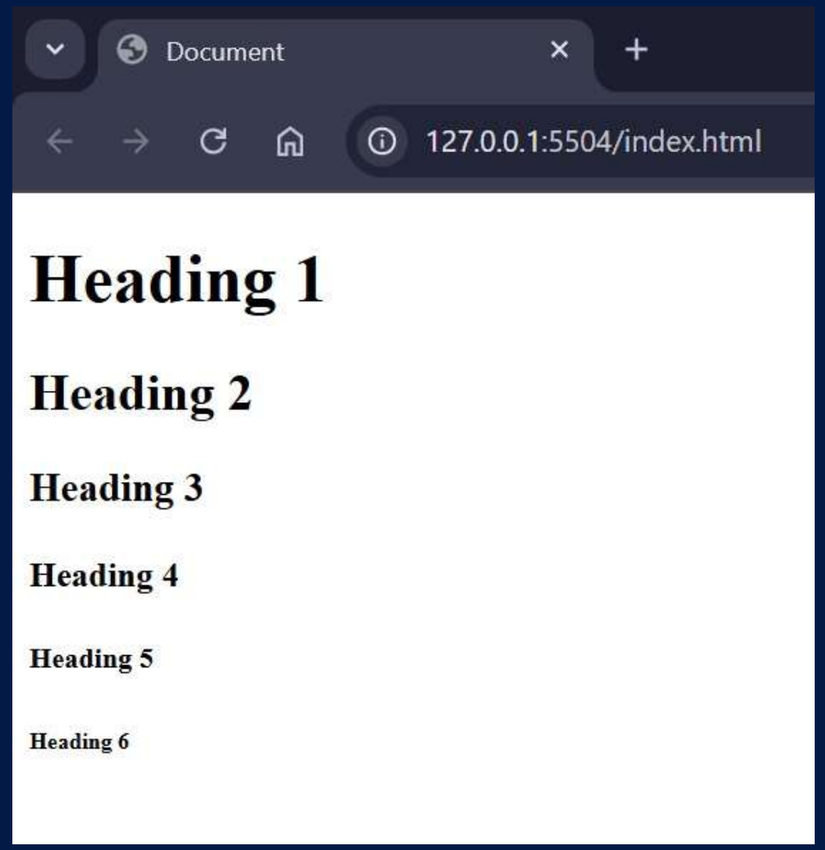
HTML-BOILER PLATE

```
<!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>

</body>
</html>
```

HTML TAGS

- **Heading tag:**

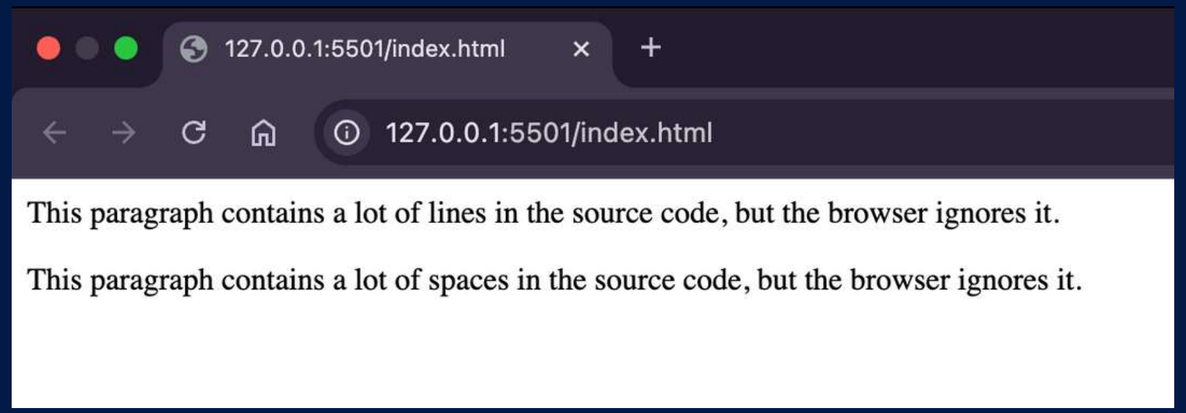
```html
<h1>Heading 1</h1>
<h2>Heading 2</h2>
<h3>Heading 3</h3>
<h4>Heading 4</h4>
<h5>Heading 5</h5>
<h6>Heading 6</h6>
```



HTML TAGS

- **Paragraph tag:**

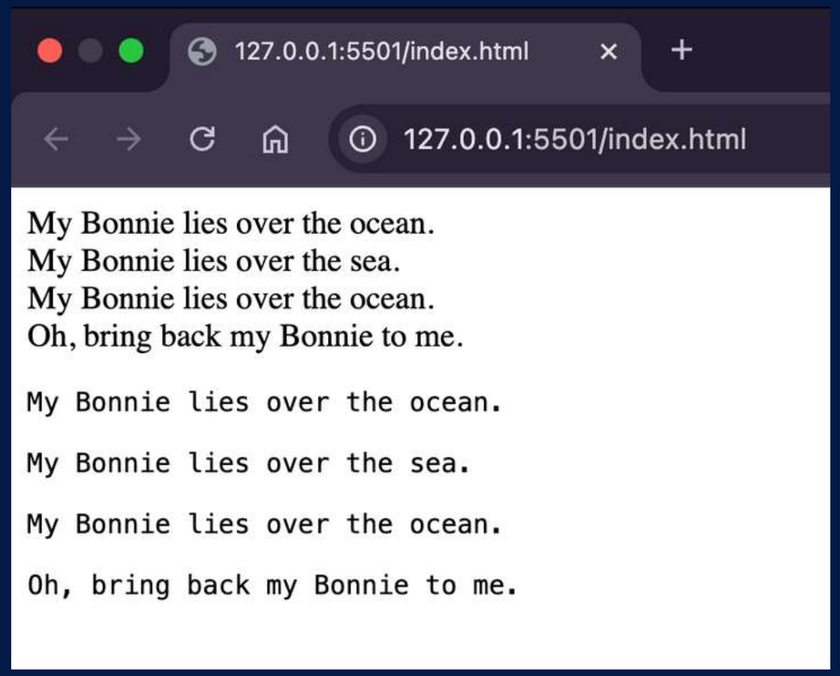
```html
<p>
    This paragraph
    contains a lot of lines
    in the source code,
    but the browser
    ignores it.
    </p>

    <p>
    This paragraph
    contains         a lot of spaces
    in the source         code,
    but the        browser
    ignores it.
    </p>
```



HTML TAGS

- **Paragraph tag:**

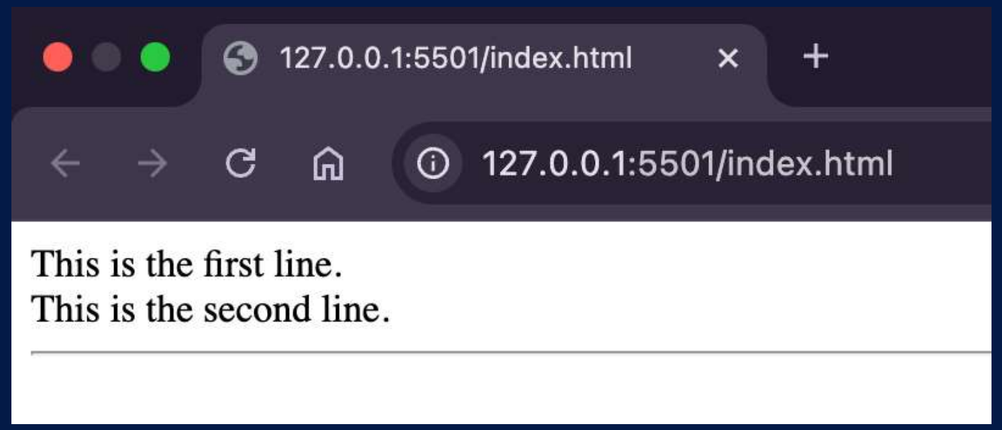
```html
    <p>
        My Bonnie lies over the ocean.
      <br>
        My Bonnie lies over the sea.
      <br>
        My Bonnie lies over the ocean.
      <br>
        Oh, bring back my Bonnie to me.
      </p>

      <pre>
My Bonnie lies over the ocean.

My Bonnie lies over the sea.

My Bonnie lies over the ocean.

Oh, bring back my Bonnie to me.
      </pre>
```



HTML TAGS

- Line Break & Horizontal Rule

```
This is the first line.<br>This is the second line.

<hr>
```



HTML TAGS

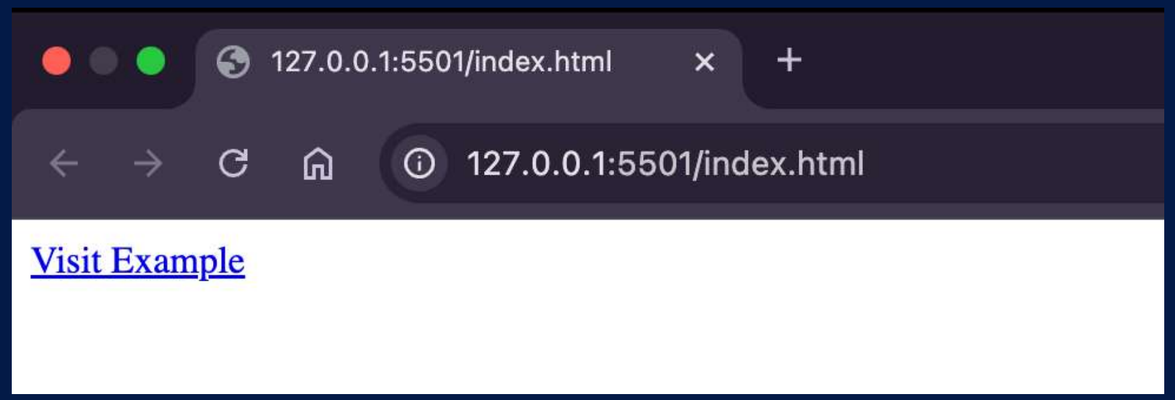
- **Formatting Text**

```html
<b>This text is bold</b>
<i>This text is italic</i>
<u>This text is underlined</u>
```

HTML TAGS

- Links and Navigation

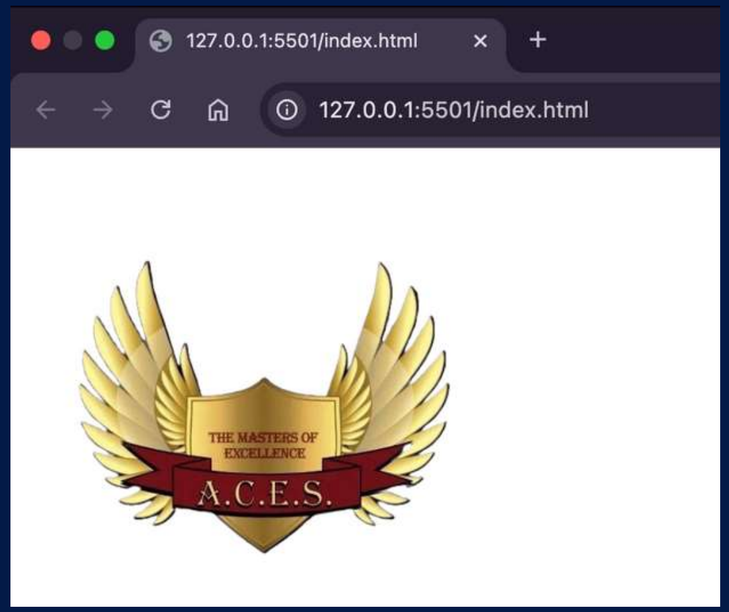
```
<a href="https://www.example.com" target="_blank">Visit Example</a>
```



HTML TAGS

- Images

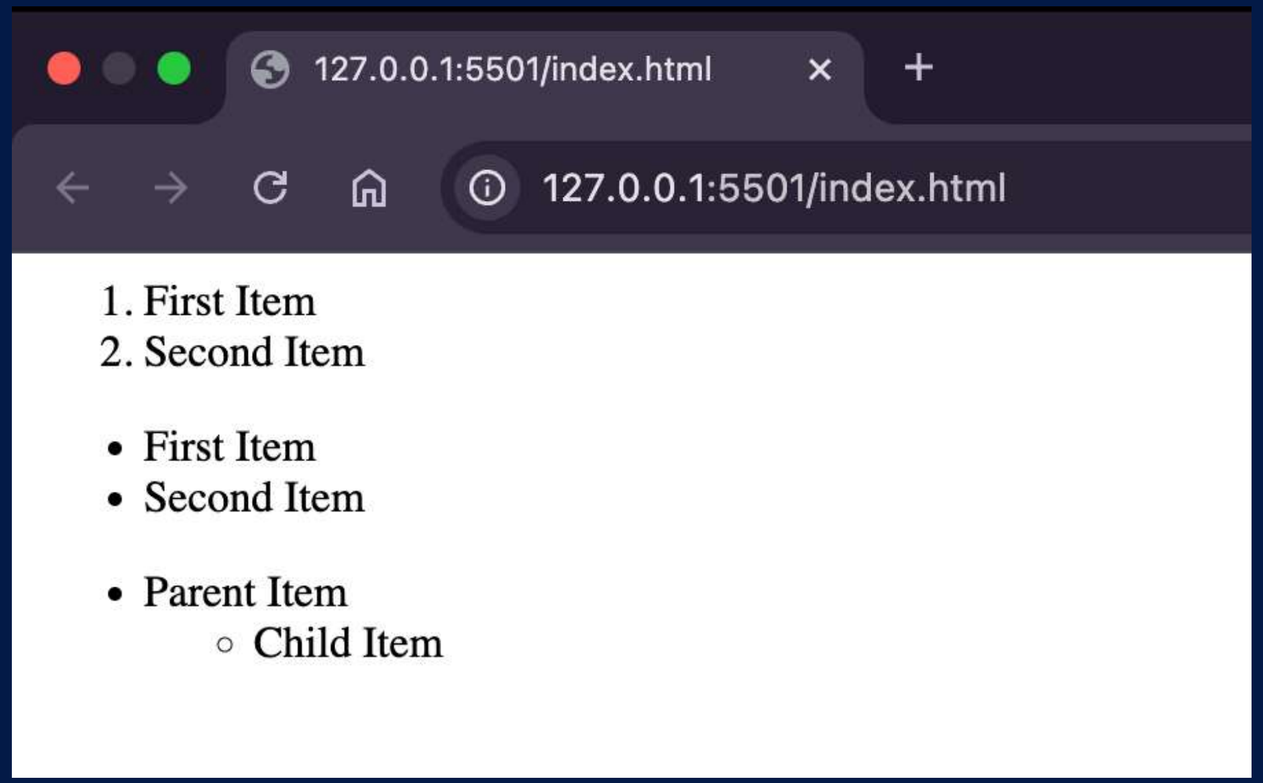
```html
<img src="https://acesdit.github.io/HackSeries/images/logo.png" alt="Description of image" width="300" height="300">
```



HTML TAGS

- Lists

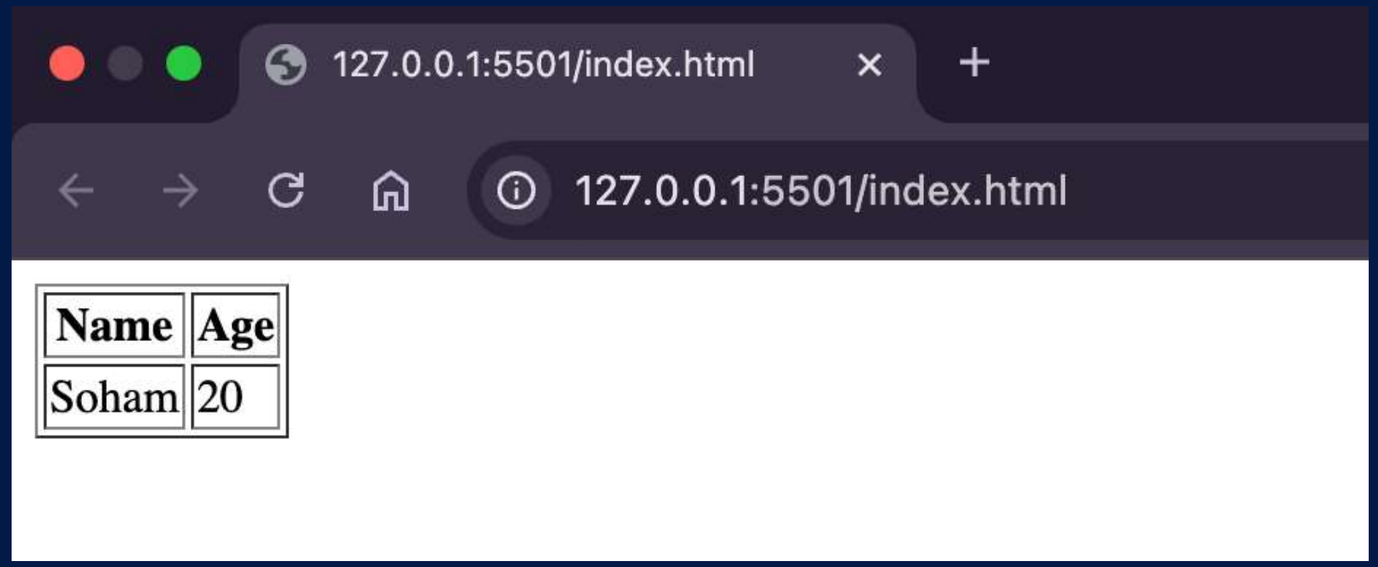
```html
<!-- ORDERED LIST -->
<ol>
    <li>First Item</li>
    <li>Second Item</li>
  </ol>


  <!-- UNORDERED LIST -->
  <ul>
    <li>First Item</li>
    <li>Second Item</li>
  </ul>


    <!-- NESTED LIST -->
  <ul>
    <li>Parent Item
      <ul>
        <li>Child Item</li>
      </ul>
    </li>
  </ul>
```



HTML TAGS

- Table

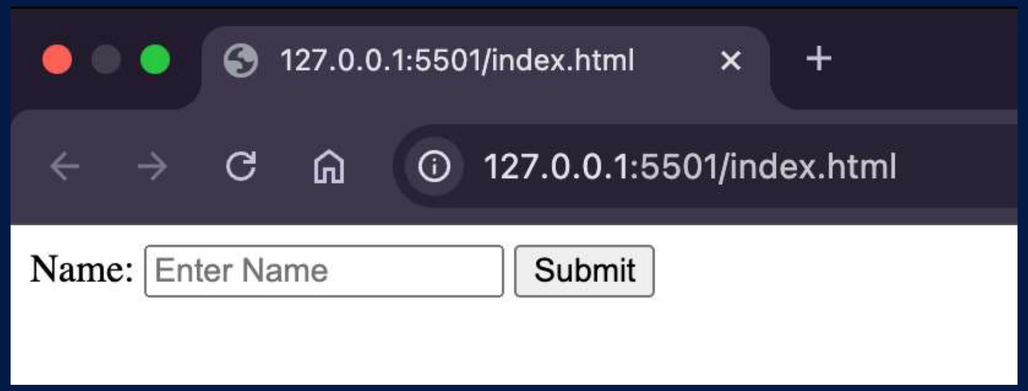
```html
<table border="1">
    <tr>
        <th>Name</th>
        <th>Age</th>
    </tr>
    <tr>
        <td>Soham</td>
        <td>20</td>
    </tr>
  </table>
```



HTML TAGS

- Forms

```html
<form action="submit.php" method="post">
    <label for="name">Name:</label>
    <input type="text" id="name" name="name" placeholder="Enter Name">
    <input type="submit" value="Submit">
  </form>
```



HTML & CSS

```html
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" href="styles.css">
</head>
<body>

<h1>This is a heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

```html
<h1 style="color: blue;">A Blue Heading</h1>

<p style="color: red;">A red paragraph.</p>
```

```html
<!DOCTYPE html>
<html>
<head>
<style>
body {background-color: rgb(255, 255, 255);}
h1   {color: blue;}
p    {color: red;}
</style>
</head>
<body>

<h1>This is a heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

```css
body {
    background-color: rgb(255, 255, 255);
  }
  h1 {
    color: blue;
  }
  p {
    color: red;
  }
```

HTML & CSS

```html
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" href="styles.css">
</head>
<body>

<h1>This is a heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

Inline

```html
<h1 style="color: blue;">A Blue Heading</h1>

<p style="color: red;">A red paragraph.</p>
```

Internal

```html
<!DOCTYPE html>
<html>
<head>
<style>
body {background-color: rgb(255, 255, 255);}
h1   {color: blue;}
p    {color: red;}
</style>
</head>
<body>

<h1>This is a heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

External

```css
body {
    background-color: rgb(255, 255, 255);
  }
  h1 {
    color: blue;
  }
  p {
    color: red;
  }
```

HTML & CSS


127.0.0.1:5502/index.html
127.0.0.1:5502/index.html
This is a heading
This is a paragraph

CSS SYNTAX

Selector

```
1  body {
2    background-color: lightblue;
3  }
```

Property

Value

CSS: SELECTORS

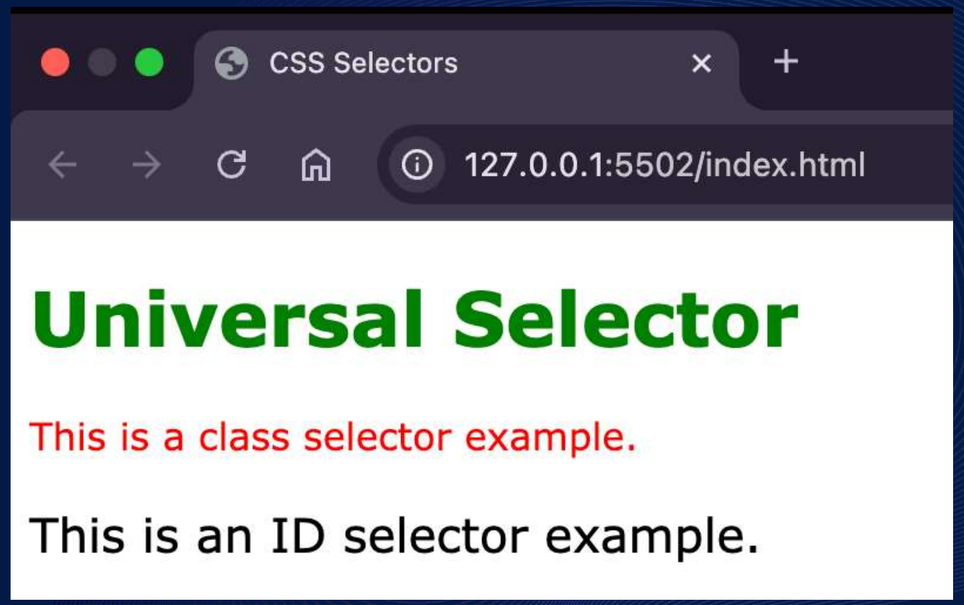
```html
<!DOCTYPE html>
<html>
<head>
  <title>CSS Selectors</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <h1>Universal Selector</h1>
  <p class="highlight">This is a class selector example.</p>
  <p id="unique">This is an ID selector example.</p>
</body>
</html>
```

styles.css > ...

```css
* {
  font-family: Verdana, sans-serif;
}
h1 {
  color: green;
}
.highlight {
  color: red;
}
#unique {
  font-size: 20px;
}
```

Selectors:

- Universal: *
- Class: **.class_name**
- ID: **#id_name**



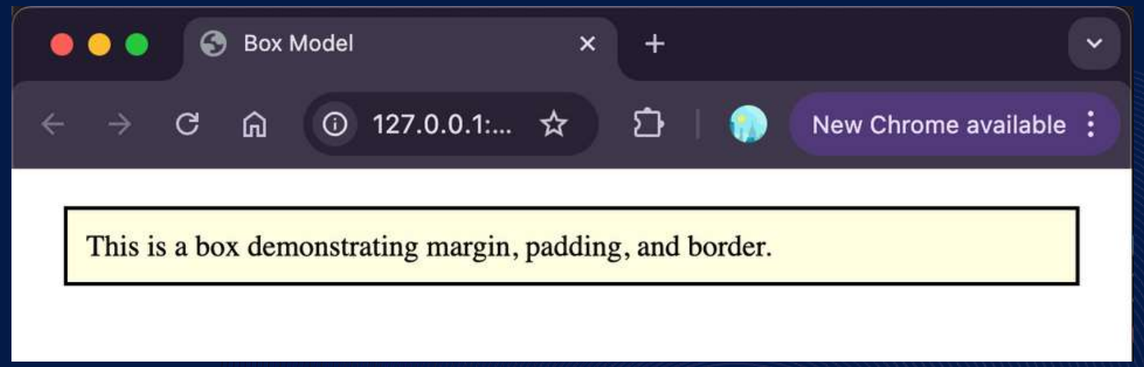
CSS: MARGINS, PADDING, AND BORDERS

index.html > ...

```html
<!DOCTYPE html>
<html>
<head>
  <title>Box Model</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <div class="box">This is a box demonstrating margin, padding, and border.</div>
</body>
</html>
```

styles.css > ...

```css
.box {
    margin: 20px;
    padding: 10px;
    border: 2px solid black;
    background-color: lightyellow;
}
```



DAY 2

CSS:FONTS AND TEXT STYLING

index.html > ...

```html
<!DOCTYPE html>
<html>
<head>
  <title>Fonts and Text Styling</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <h1>Styled Header</h1>
  <p>This paragraph demonstrates text styling with fonts and decorations.</p>
</body>
</html>
```

styles.css > ...

```css
h1 {
    font-family: 'Courier New', monospace;
    font-size: 24px;
    text-align: center;
  }
  p {
    line-height: 1.6;
    text-decoration: underline;
  }
```

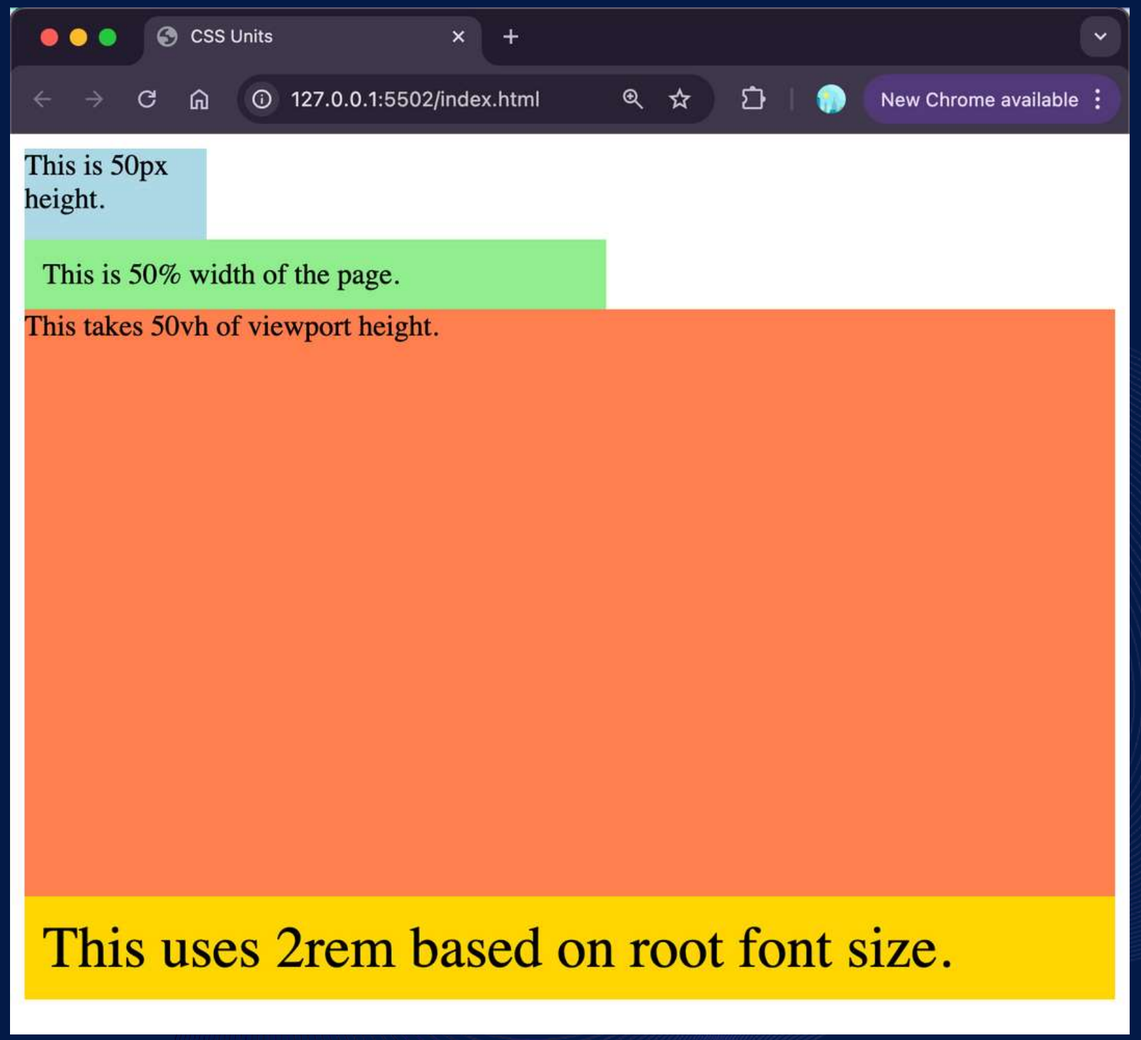
CSS: UNITS

index.html > ...

```html
<!DOCTYPE html>
<html>
<head>
  <title>CSS Units</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <div class="px-unit">This is 50px height.</div>
  <div class="percent-unit">This is 50% width of the page.</div>
  <div class="vh-unit">This takes 50vh of viewport height.</div>
  <div class="rem-unit">This uses 2rem based on root font size.</div>
</body>
</html>
```

styles.css > ...

```css
/* Root font size for rem */
html {
  font-size: 16px;
}

.px-unit {
  height: 50px;
  width: 100px;
  background-color: lightblue;
}

.percent-unit {
  width: 50%;
  background-color: lightgreen;
  padding: 10px;
}

.vh-unit {
  height: 50vh;
  background-color: coral;
}

.rem-unit {
  font-size: 2rem; /* 2 * 16px = 32px */
  padding: 10px;
  background-color: gold;
}
```



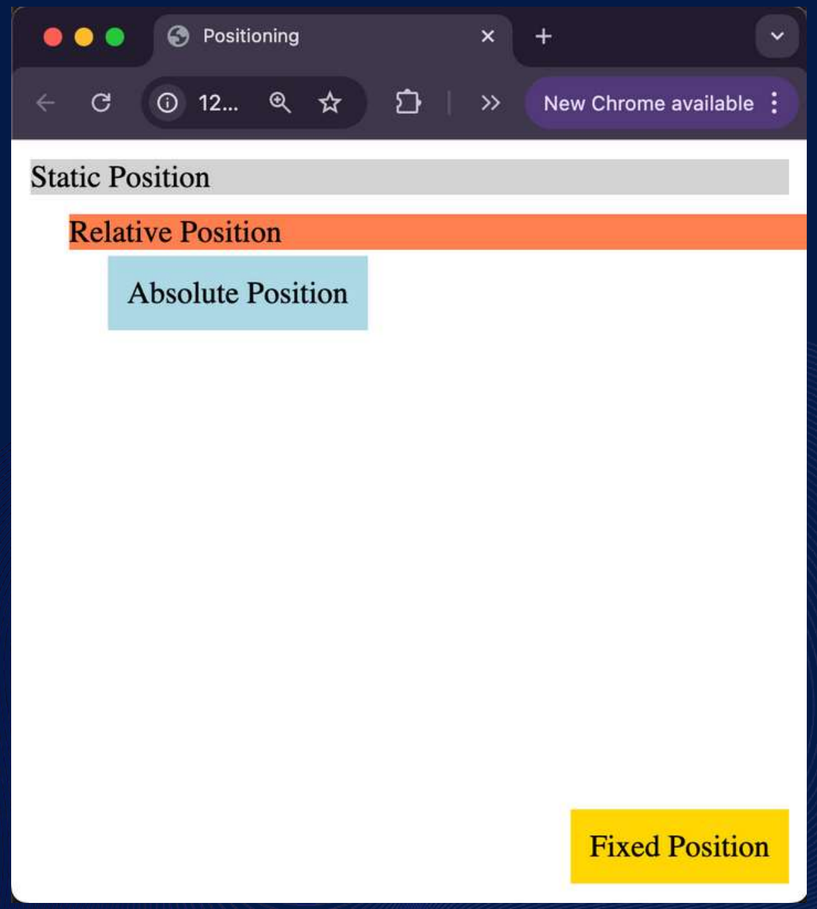
CSS: LAYOUTS- POSITIONING

index.html > ...

```html
<!DOCTYPE html>
<html>
<head>
  <title>Positioning</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <div class="static">Static Position</div>
  <div class="relative">Relative Position</div>
  <div class="absolute">Absolute Position</div>
  <div class="fixed">Fixed Position</div>
</body>
</html>
```

styles.css > .fixed

```css
body {
  position: relative;
  margin: 0;
}

.static {
  position: static;
  background-color: lightgray;
  margin: 10px;
}

.relative {
  position: relative;
  left: 20px;
  background-color: coral;
  margin: 10px;
}

.absolute {
  position: absolute;
  top: 50px;
  left: 50px;
  background-color: lightblue;
  padding: 10px;
}

.fixed {
  position: fixed;
  bottom: 10px;
  right: 10px;
  background-color: gold;
  padding: 10px;
}
```



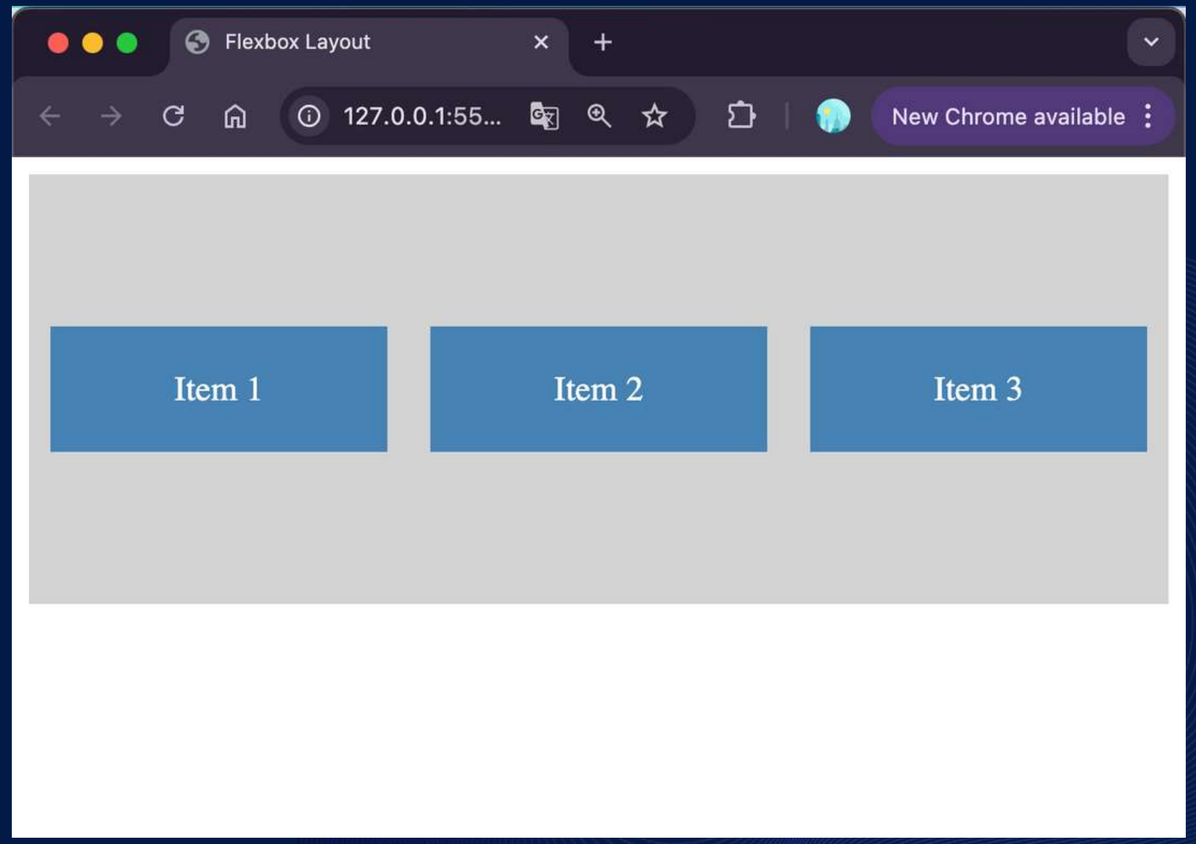
CSS: LAYOUTS- FLEX

index.html > ...

```html
<!DOCTYPE html>
<html>
<head>
  <title>Flexbox Layout</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <div class="flex-container">
    <div class="flex-item">Item 1</div>
    <div class="flex-item">Item 2</div>
    <div class="flex-item">Item 3</div>
  </div>
</body>
</html>
```

styles.css > ...

```css
/* Flex Container */
.flex-container {
  display: flex;
  flex-direction: row; /* Horizontal layout */
  flex-wrap: wrap; /* Items will wrap if needed */
  justify-content: space-around;
  align-items: center;
  height: 200px;
  background-color: lightgray;
}

/* Flex Items */
.flex-item {
  flex: 1; /* Equal flex basis for all items */
  margin: 10px;
  padding: 20px;
  background-color: steelblue;
  color: white;
  text-align: center;
}
```



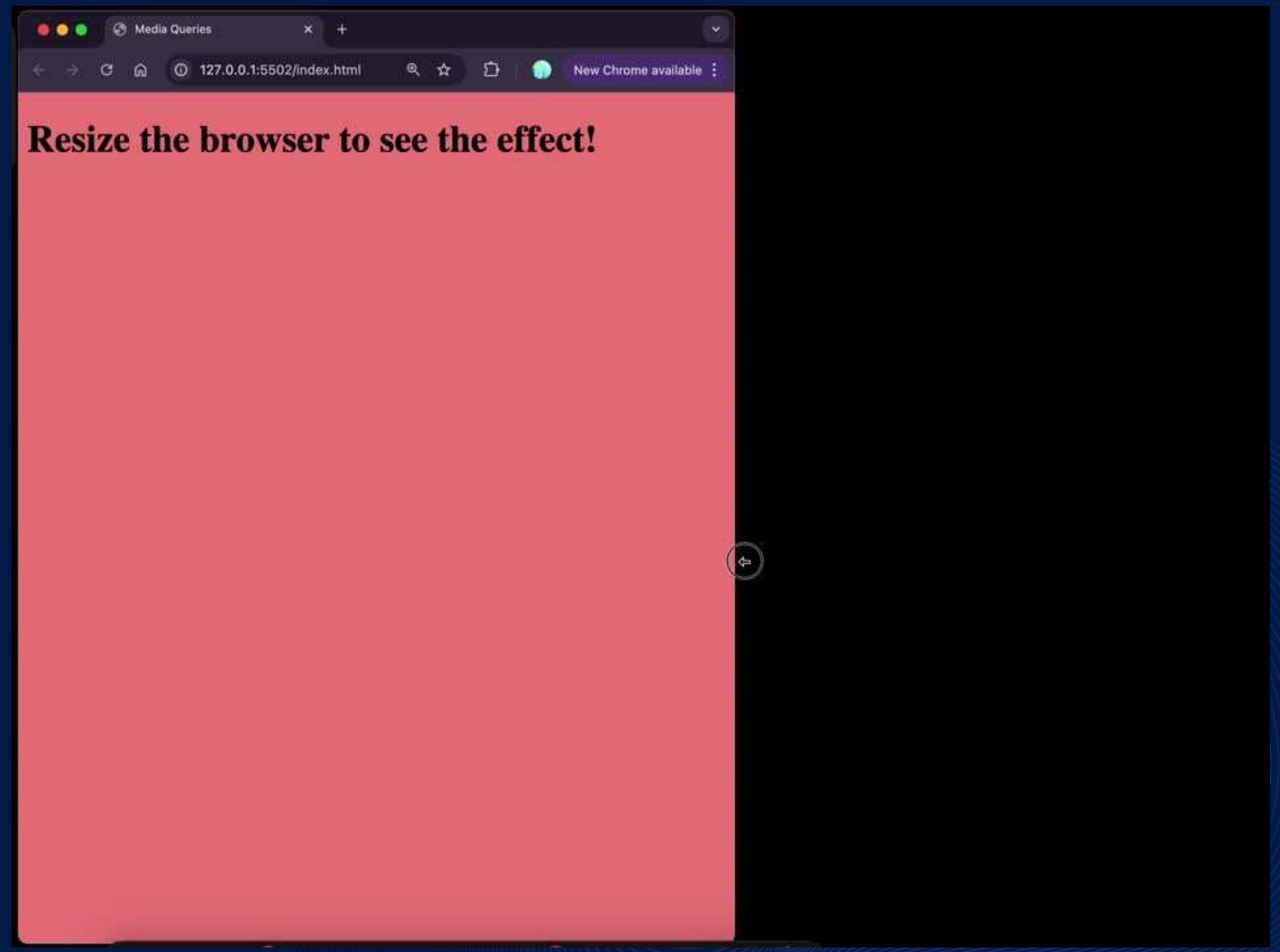
CSS: MEDIA QUERIES

index.html > ...

```html
<!DOCTYPE html>
<html>
<head>
  <title>Media Queries</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <h1>Resize the browser to see the effect!</h1>
</body>
</html>
```

styles.css > ...

```css
body {
    background-color: lightblue;
  }
  @media (max-width: 768px) {
    body {
      background-color: lightcoral;
    }
  }
```



- max-height, min-width, min-height

BEST PRACTICES

ACES

- .vscode
- Hackseries
- index.html
- styles.css

index.html > ...

```html
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <h1>External CSS is the Best Practice!</h1>
  <p>Organize your stylesheets for better maintainability.</p>
</body>
</html>
```

styles.css

styles.css > ...

```css
.container {
  display: flex;
  justify-content: center;
  align-items: center;
  height: 100vh;
  background-color: lightgray;
}
.item {
  padding: 20px;
  background-color: steelblue;
  color: white;
  margin: 10px;
}
```

DAY 4

JAVASCRIPT

JS & HTML

index.html > html

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>JavaScript Demo</title>
</head>
<body>
  <h1>JavaScript Demo</h1>
  <p>Open the browser console to see the output.</p>

  <!-- Link to your external script -->
  <script src="script.js"></script>
</body>
</html>
```

JS- WORD VS KEYWORD

Aspect	word	keyword
Definition	Any string or identifier in JavaScript.	Reserved words with specific meanings.
purpose	Used as names for variables, functions, etc.	Used to define JavaScript syntax or logic.
example	myVar, calculate, userName	if, else, let, const, return
restriction	Can be user-defined and custom.	Cannot be used as identifiers or variables.

JS- VAR. LET. CONST

JS script.js > ...

```js
var x = 10; // Can be redeclared
x = 20;
console.log(x); // Output: 20

let y = 10; // Block scoped
y = 20;
console.log(y); // Output: 20

const z = 10; // Cannot be reassigned
console.log(z); // Output: 10
```

Elements Console Sources

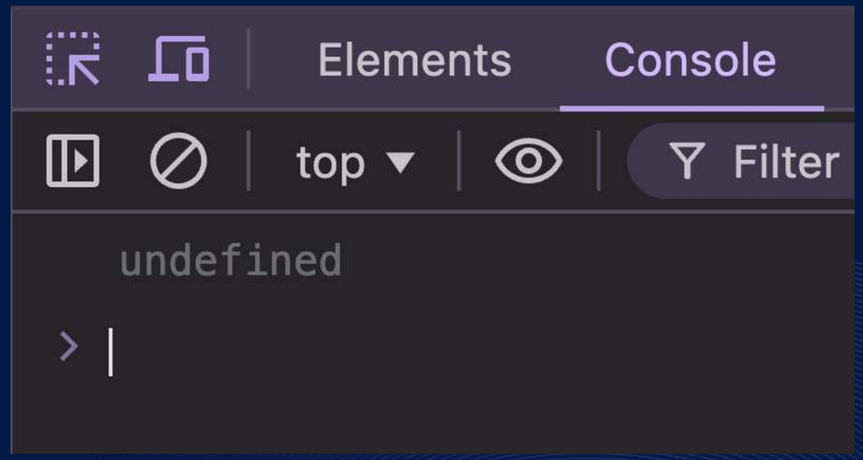
top ▾ Filter

```
20
20
10
>
```

JS- HOISTING

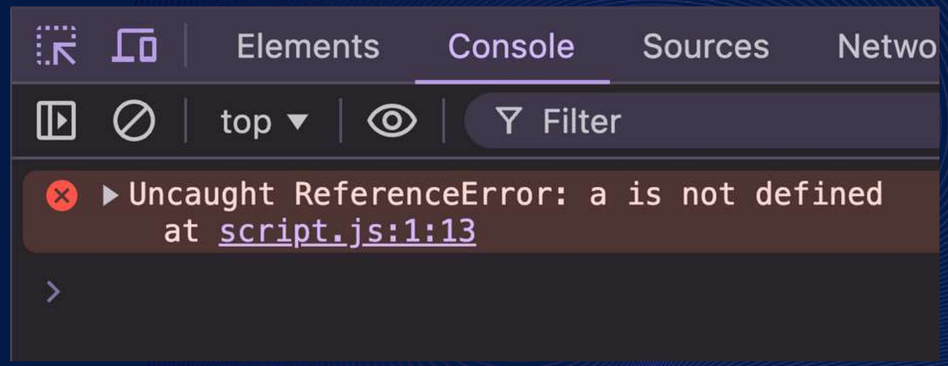
JS script.js > ...

```
console.log(a); // Output: undefined
var a = 5;
```



JS script.js > ...

```
// Hoisting with var
console.log(x); // Output: undefined (hoisted)
var x = 5;
console.log(x); // Output: 5

// Hoisting with let
// Uncommenting the line below will throw an error
console.log(y);
let y = 10;
console.log(y); // Output: 10
```



JS- TYPES

- Primitive
- Referenced

JS script.js > ...

```js
// Primitive Types
let str = "Hello";
let num = 42;
let bool = true;
let undef;
let n = null;

// Reference Types
let array = [1, 2, 3];
let obj = { name: "John", age: 30 };

console.log(typeof str);  // Output: string
console.log(typeof obj);  // Output: object
```

Elements Console

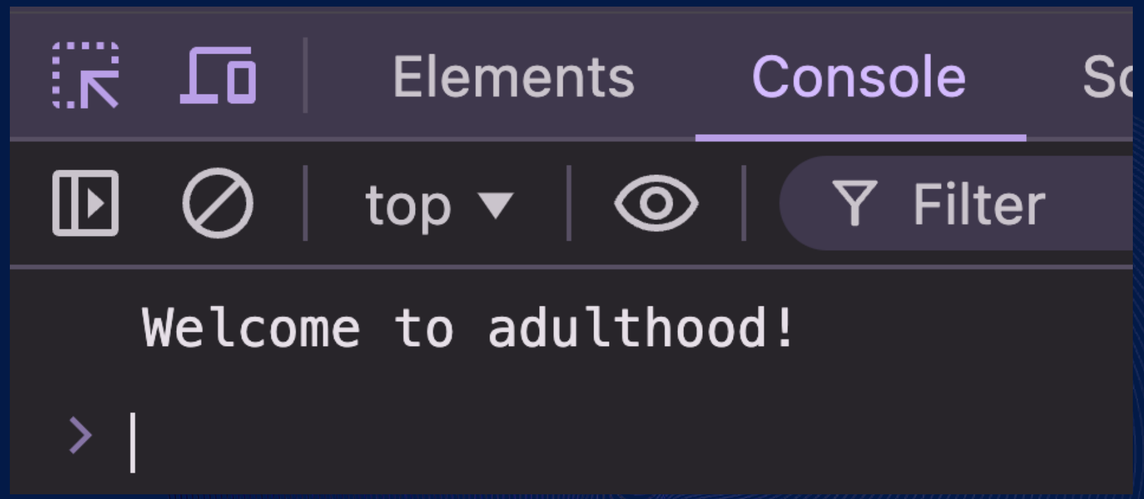
top Filter

```
string
object
>
```

JS- CONDITIONALS

- if
- else
- else if

```js
let age = 18;

if (age < 18) {
  console.log("You are a minor.");
} else if (age === 18) {
  console.log("Welcome to adulthood!"); // Output: Welcome to adulthood!
} else {
  console.log("You are an adult.");
}
```



JS-LOOPS

- while
- for

JS script.js > ...

```js
console.log("While loop");
let i = 0;
while (i < 3) {
  console.log(i); // Output: 0, 1, 2
  i++;
}

console.log("For loop");
for (let j = 0; j < 3; j++) {
  console.log(j); // Output: 0, 1, 2
}
```

Elements Console

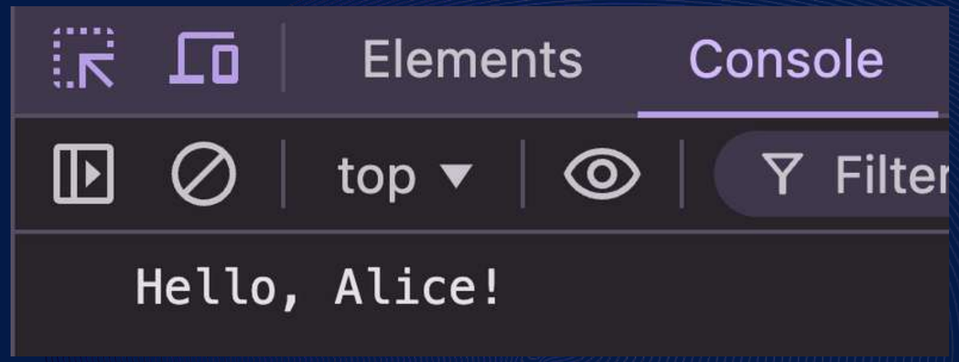
top Filter

```
While loop
0
1
2
For loop
0
1
2
```

JS-FUNCTIONS

JS script.js > ...

```javascript
function sayHello(name) {
    console.log("Hello, " + name + "!"); // Output: Hello, Alice!
}

sayHello("Alice");
```



JS- ARRAY

JS script.js > ...

```javascript
let fruits = ["apple", "banana", "cherry"];

// Add element
fruits.push("date");
console.log(fruits);

// Remove element
fruits.pop();
console.log(fruits);

// Access element
console.log(fruits[1]);
```

Elements Console Sources Network

top Filter

```
▶ (4) ['apple', 'banana', 'cherry', 'date']
▶ (3) ['apple', 'banana', 'cherry']
banana
>
```

JS-OBJECT

JS script.js > ...

```
1  let person = {
2      name: "Alice",
3      age: 25
4  };
5
6  console.log(person.name);
7
```

Elements Console

top Filter

```
Alice
>
```

JS-DOM

Document Object Model

Four pillars of DOM:

1) Selection of a element
2) Changing HTML
3) Changing CSS
4)Event Listener

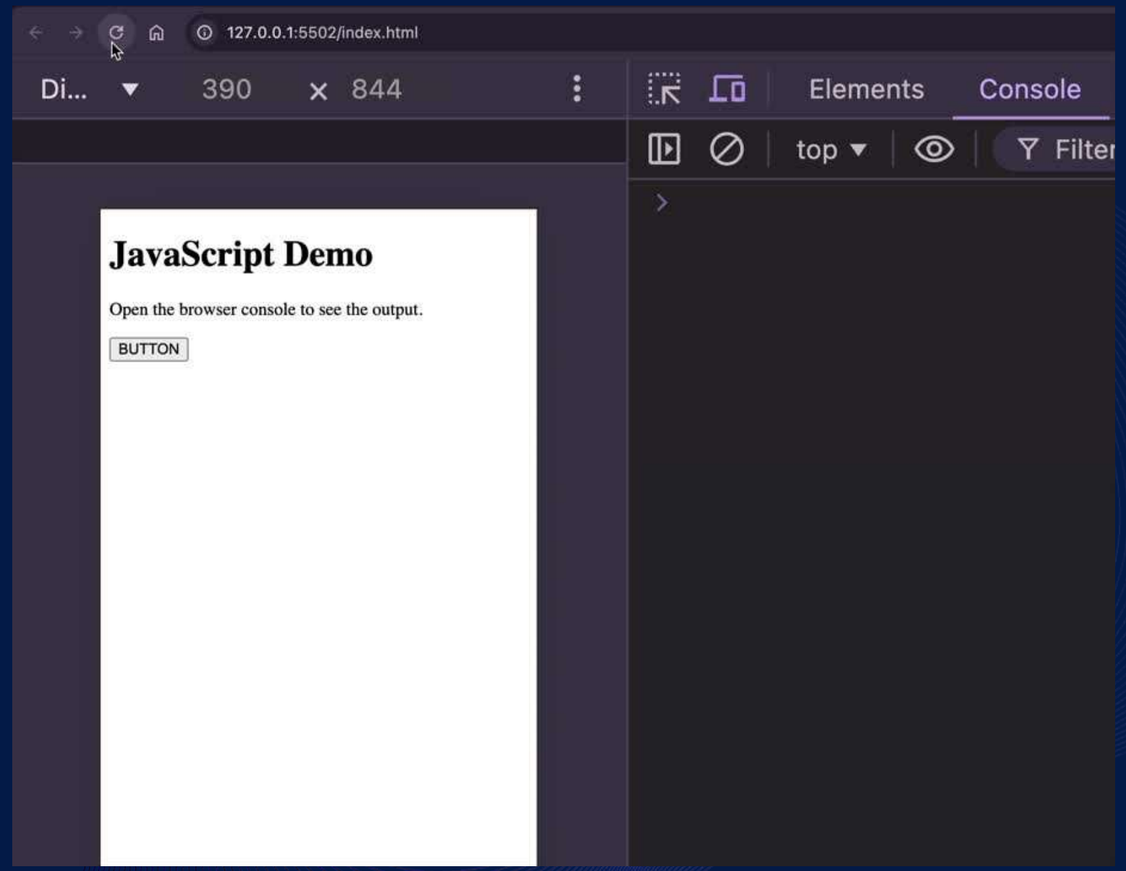
DOM- SELECTION OF A ELEMENT

Purpose: To identify and select specific elements in the DOM for manipulation.

Types:

- By ID: document.getElementById()
- By Class: document.getElementsByClassName()
- By Tag Name: document.getElementsByTagName()
- By CSS Selector: document.querySelector() and document.querySelectorAll()

```
JS script.js > ...
// Select the button element
var button = document.getElementById("btn");

// Use setTimeout to change the color after 2 seconds
setTimeout(function() {
  button.style.backgroundColor = "red"; // Change the background color to red
}, 2000);
```



DOM- CHANGING HTML

Purpose: Modify the structure or content of the selected elements..

Types:

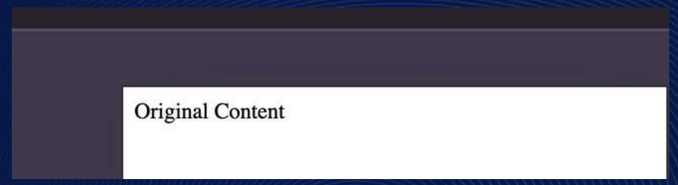
- Changing inner content: .innerHTML or .textContent
- Changing attributes: .setAttribute() or .removeAttribute()

index.html > ...

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Test DOM</title>
</head>
<body>
  <div id="header">Original Content</div>
  <script src="script.js"></script> <!-- Link to your external JS file -->
</body>
</html>
```

JS script.js > ...

```js
// Change the inner HTML
let element = document.getElementById('header');
element.innerHTML = "<h1>Welcome to the DOM World!</h1>";

// Change the text content
element.textContent = "Hello DOM!";
```



DOM- CHANGING CSS

Purpose: Alter the styling of selected elements..

Types:

- Inline CSS: Modify styles using .style
- Add/Remove Classes: Use .classList.add(), .classList.remove(), .classList.toggle()

index.html > html

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Test CSS Manipulation</title>
  <style>
    .highlight {
      background-color: yellow;
      font-weight: bold;
    }
    .active {
      text-decoration: underline;
    }
  </style>
</head>
<body>
  <h1 id="header">CSS Manipulation Test</h1>

  <!-- Toggle Buttons -->
  <button onclick="applyStyles()">Apply Styles</button>
  <button onclick="addClass()">Add Class</button>
  <button onclick="removeClass()">Remove Class</button>
  <button onclick="toggleClass()">Toggle Class</button>

  <script src="script.js"></script>
</body>
</html>
```

script.js > ...

```javascript
let element = document.getElementById('header');

// Apply inline styles
function applyStyles() {
  element.style.color = 'blue';
  element.style.fontSize = '24px';
}

// Add a class
function addClass() {
  element.classList.add('highlight');
}

// Remove a class
function removeClass() {
  element.classList.remove('highlight');
}

// Toggle a class
function toggleClass() {
  element.classList.toggle('active');
}
```

DOM- EVENT LISTENER

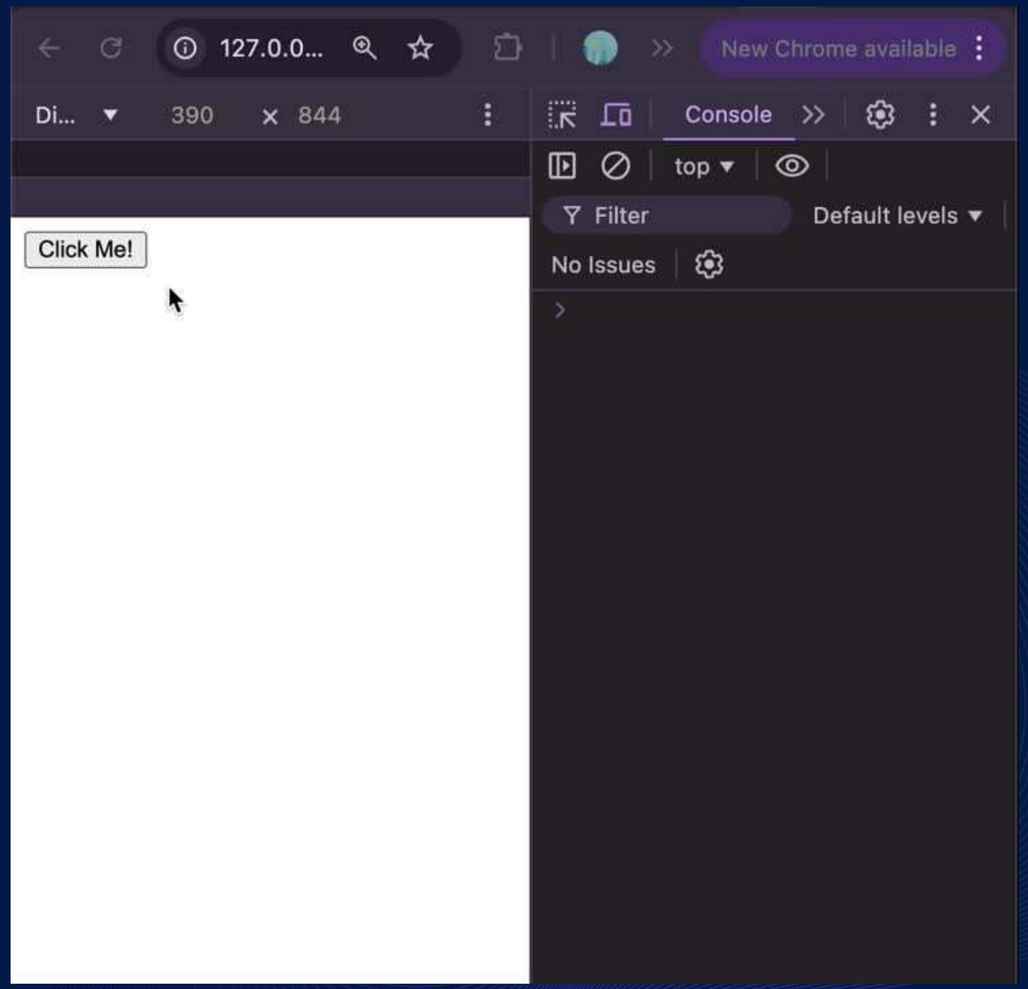
Purpose: Respond to user interactions or other events..

Types:

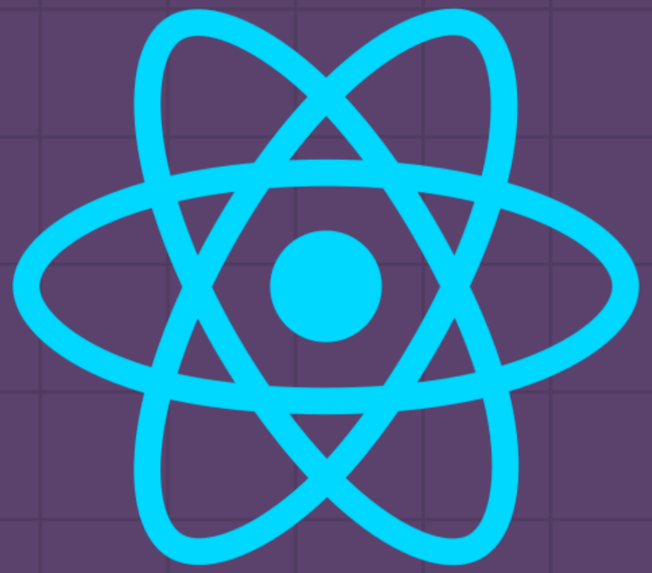
- Mouse Events: click, mouseover, mouseout
- Keyboard Events: keydown, keyup
- Form Events: submit, change, focus
- Other Events: load, resize

JS script.js > ...

```javascript
// Add a click event listener
let button = document.querySelector('.btn');
button.onclick = function () {
  alert('Button clicked!');
};

// Add a keydown event listener
document.addEventListener('keydown', function (event) {
  console.log(`Key pressed: ${event.key}`);
});
```



Hackseries 01



React JS

Introduction to React

1. What is React?
2. Working of DOM
3. Problems with JS
4. Working of React
5. JS Vs React
6. Intro to Components

- HTML is required for React
- CSS is required for React
- JS is required for React

1. What is React

1. JavaScript library to build Dynamic and interactive user interfaces
2. Developed at Facebook in 2011.
3. Currently most widely used JS library for front-end development.
4. Used to create single page application (page does not re-load).

2. Working of DOM

1. Browser takes HTML and create DOM.
2. JS helps us modify DOM based on user actions or events.
3. In big applications, Working with DOM becomes complicated

3. Problems with JavaScript

1. React has a simpler mental model
2. JS is cumbersome
3. JS is Error-prone
4. JS is Hard to maintain

{.js}
JavaScript

4. Working of React

1. No need to worry about querying and updating DOM elements.
2. React creates a web page with small and reusable components
3. React will take care of creating and updating DOM elements.
4. IT saves a lot of time, cheezein aasan hai, pahele se likhi hui hain

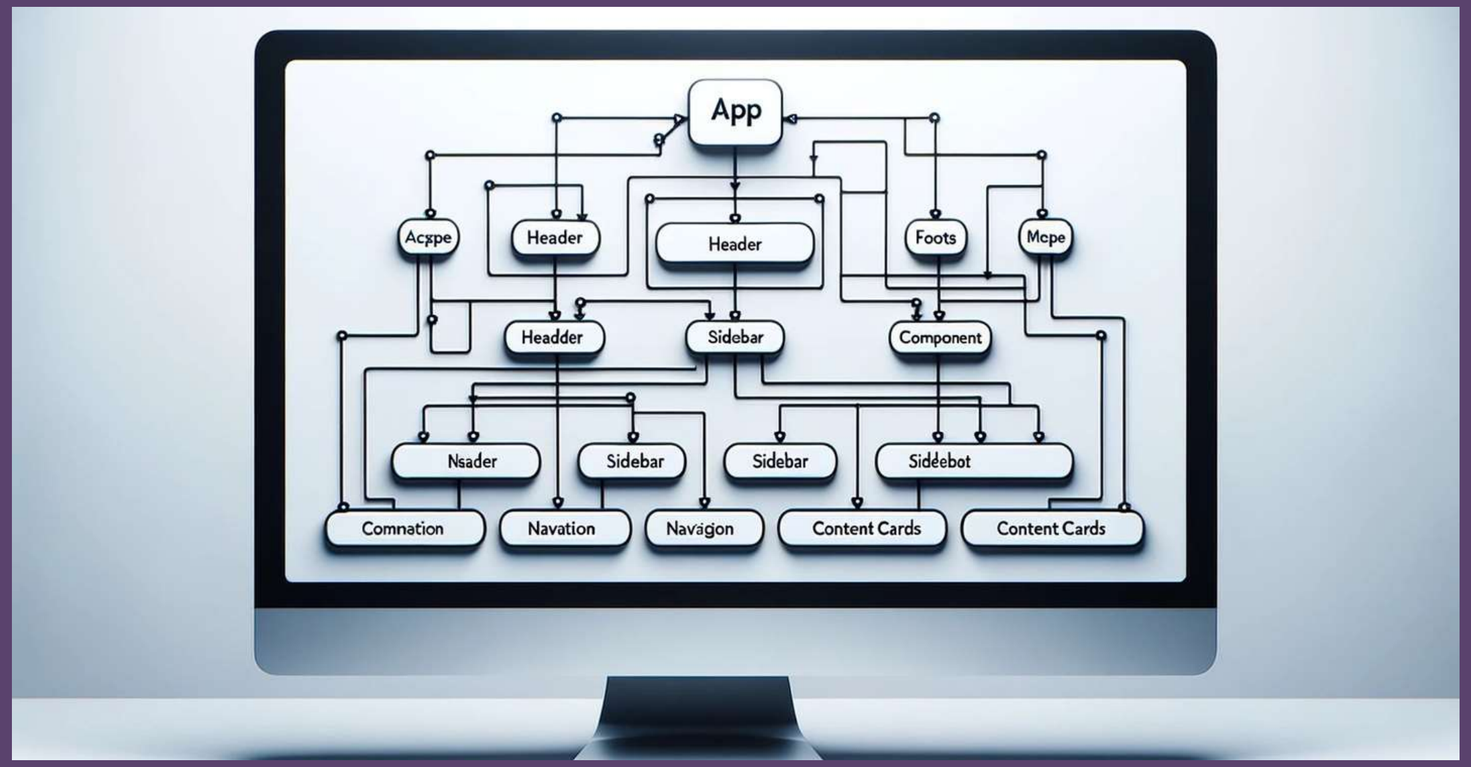
5. JS Vs React

1. **JS is imperative:** You define steps to reach your desired state.
2. **React is Declarative:** You define the target UI state and then react figures out how to reach that state.

6. Introduction to Components

Components help us write reusable, modular and better organized code.

6. Introduction to Components



- React application is a tree of components with App Component as the root bringing everything together.

Introduction Revision

1. What is React?
2. Working of DOM
3. Problems with JS
4. Working of React
5. JS Vs React
6. Intro to Components

Create a React App

1. SETUP IDE
2. CREATE A REACT APP
3. PROJECT STRUCTURE

7. What is IDE

1. IDE stands for Integrated Development Environment.
2. Software suite that consolidates basic tools

required for software development.

1. Central hub for coding, finding problems, and testing.
1. Designed to improve developer efficiency.

7. Need of IDE

1. Streamlines development.
2. Increases productivity.
3. Simplifies complex tasks.
4. Offers a unified workspace.
5. IDE Features:
    1. Code Autocomplete
    2. Syntax Highlighting
    3. Version Control
    4. Error Checking

7. Install latest Node

1. Search Download NodeJS

7. Installation & Setup

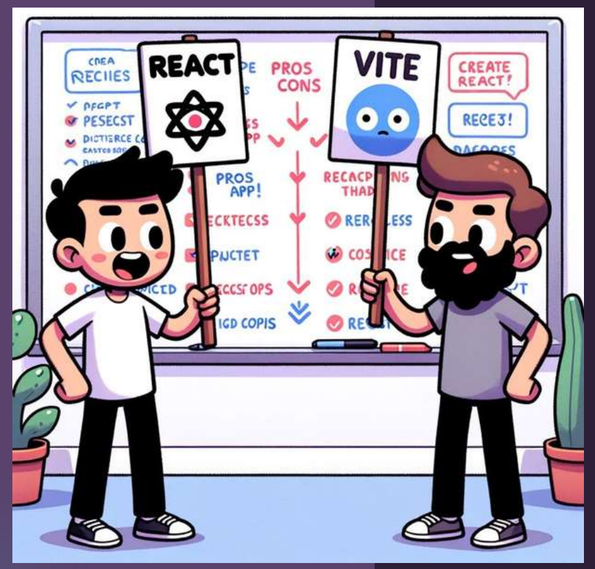
1. Search VS Code
2. Keep Your Software up to date

7. VsCode Extensions and Settings

1. Live Server / Live Preview
2. Prettier (Format on Save)
3. Line Wrap
4. Tab Size from 4 to 2

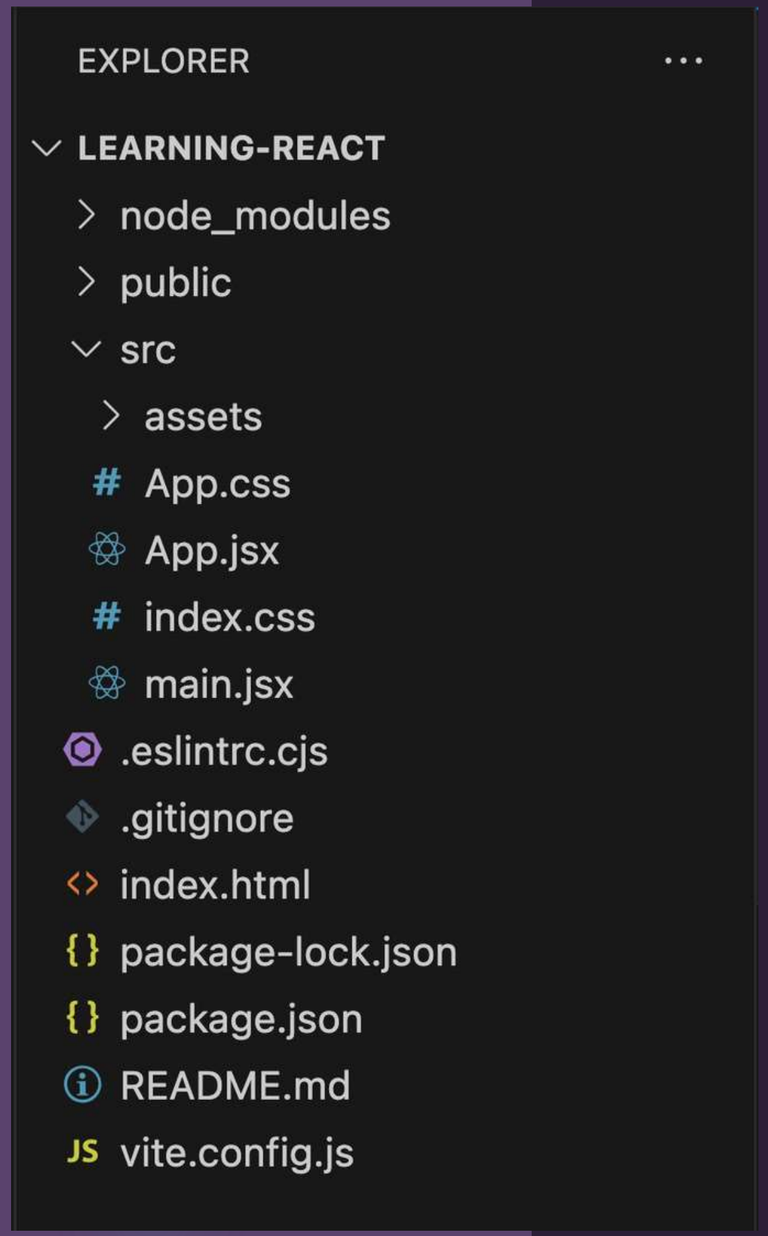
8. Create a React App

1. Official tool is CRA(Create React APP)
2. Vite is a modern tool to create React Project.
3. Vite produces Quick and Small bundle size.
4. Vite: Use *npm run dev* to launch dev server.
5. Use *npm start* for CRA.



9. Project Structure

1. node_modules/ has all the installed node packages
2. public/ Directory: Contains static files that don't change.
3. src/ Directory: Main folder for the React code.
    1. components/: Reusable parts of the UI, like buttons or headers.
    2. assets/: Images, fonts, and other static files.
    3. styles/: CSS or stylesheets.
4. package.json contains information about this project like name, version, dependencies on other react packages.
5. vite.config.js contains vite config.



Creating React Components

1. File Extensions
2. Class vs Function Components
3. What is JSX?
4. Exporting component
5. Other important Points
6. Dynamic Components
7. Reusable Components

10. File Extensions

.JS

- Stands for JavaScript
- Contains regular JavaScript code
- Used for general logic and components

.JSX

- Stands for JavaScript XML
- Combines JavaScript with HTML-like tags
- Makes it easier to design UI components

Class vs Function Components

Class Components

- Stateful: Can manage state.
- Lifecycle: Access to lifecycle methods.
- Verbose: More boilerplate code.
- Not Preferred anymore.

Functional Components

- Initially stateless.
- Can use Hooks for state and effects.
- Simpler and more concise.
- More Popular.

What is JSX?

1. Definition: JSX determines how the UI will look wherever the component is used.
2. Not HTML: Though it resembles HTML, you're actually writing JSX, which stands for JavaScript XML.
3. Conversion: JSX gets converted to regular JavaScript.
4. Babeljs.io/repl is a tool that allows you to see how JSX is transformed into JavaScript.

Exporting components

1. Enables the use of a component in other parts.
2. Default Export: Allows exporting a single component as the default from a module.
3. Named Export: Allows exporting multiple items from a module.
4. Importing: To use an exported component, you need to import it in the destination file using import syntax.

Component.js

```
export default
function
Button() {
  ...
}
```

one default export

Components.js

```
export function
Slider() {
  ...
}
```

```
export function
Checkbox() {
  ...
}
```

multiple named exports

MixedComponents.js

```
export function
Avatar() {
  ...
}
```

```
export default
function
FriendsList() {
  ...
}
```

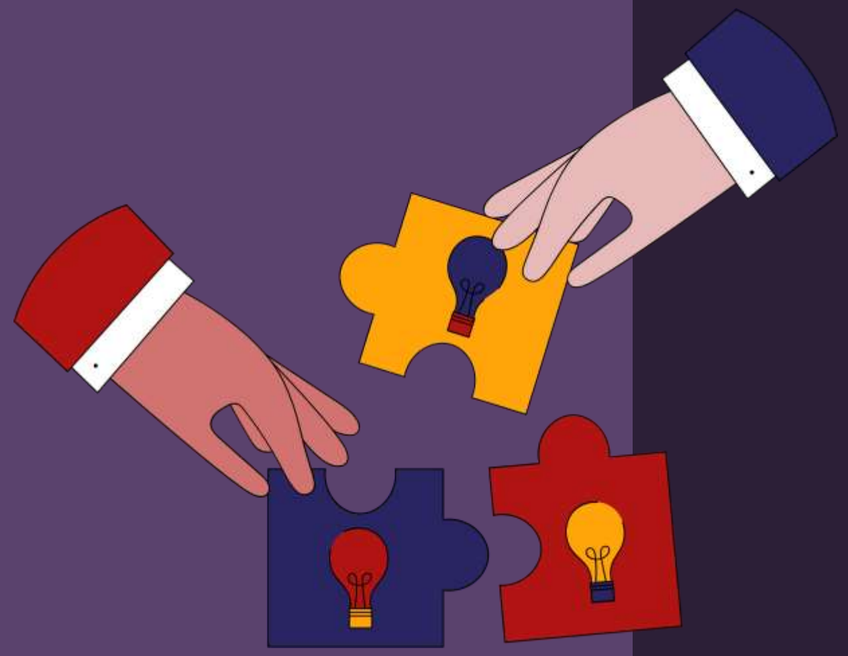
named export(s)
and one default export

Other important Points

1. Naming: Must be capitalized; lowercase for default HTML.
2. HTML: Unlike vanilla JS where you can't directly write HTML, in React, you can embed HTML-like syntax using

JSX.

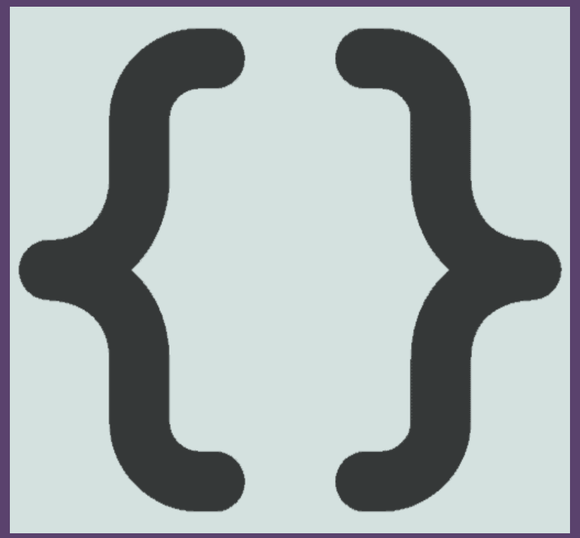
1. CSS: In React, CSS can be directly imported into component files, allowing for modular and component-specific styling.



Dynamic Components

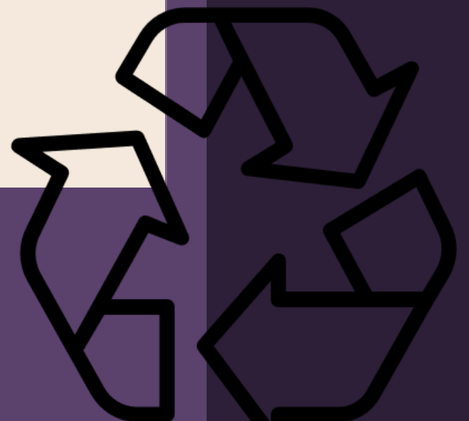
1. Dynamic Content: JSX allows the creation of dynamic and interactive UI components.
2. JavaScript Expressions: Using {}, we can embed any JS expression

directly within JSX. This includes variables, function calls, and more.



Reusable Components

1. Modularity: Components are modular, allowing for easy reuse across different parts of an application.
2. Consistency: Reusing components ensures UI consistency and reduces the chance of discrepancies.
3. Efficiency: Reduces development time and effort by avoiding duplication of code.
4. Maintainability: Changes made to a reused component reflect everywhere it's used, simplifying updates and bug fixes.



4 Day ago

Post One

Croque monsieur paneer cheese triangles. When the cheese comes out everybody's happy cheeseburger melted cheese pepper jack croque

7	3224	21
Reads	Views	Comments

1 week ago

Post Two

Croque monsieur paneer cheese triangles. When the cheese comes out everybody's happy cheeseburger melted cheese pepper jack croque

11	1699	27
Reads	Views	Comments

4 week ago

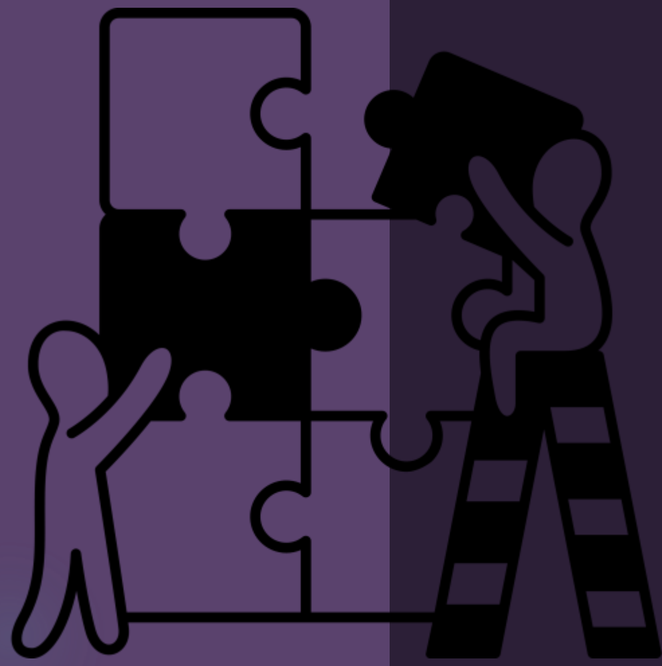
Post Three

Croque monsieur paneer cheese triangles. When the cheese comes out everybody's happy cheeseburger melted cheese pepper jack croque

4	1624	17
Reads	Views	Comments

Creating React Components Revision

1. File Extensions
2. Class vs Function Components
3. What is JSX?
4. Exporting component
5. Other important Points
6. Dynamic Components
7. Reusable Components



Including Bootstrap

1. Responsive: Mobile-first design for all device sizes.
2. Components: Pre-styled elements like buttons and navbars.
3. Customizable: Modify default styles as needed.
4. Cross-Browser: Consistent look across browsers.
5. Open-Source: Free with community support.

1. Install:

npm i bootstrap@5.3.2

1. import

```
import "bootstrap/dist/css/bootstrap.min.css";
```

Project: Clock

Bharat Clock

This is the clock that shows the time in Bharat at all times

This is the current time: 26/10/2023 - 10:38:17 AM

Fragments

1. What?

Allows grouping of multiple elements without extra DOM nodes.

1. Why?

- Return multiple elements without a wrapping parent.
- Cleaner DOM and consistent styling.

2. How? Two syntaxes:

3. <React.Fragment>...</React.Fragment>
4. Short: <>...</>

Map Method

1. Purpose: Render lists from array data.
2. JSX Elements: Transform array items into JSX.
3. Inline Rendering: Directly inside JSX

```
{items.map(item => <li
    key={item.id}>{item.name}</li>)}
```

1. Key Prop: Assign unique key for optimized re-renders.

```
<div key={item.id}>{item.name}</div>
```

Conditional Rendering

Conditional Rendering

- Displaying content based on certain conditions.
- Allows for dynamic user interfaces. Methods
  - If-else statements: Choose between two blocks of content.
  - Ternary operators: Quick way to choose between two options.
  - Logical operators: Useful for rendering content when a condition is true.

Benefits

- Enhances user experience.
- Reduces unnecessary rendering.
- Makes apps more interactive and responsive.

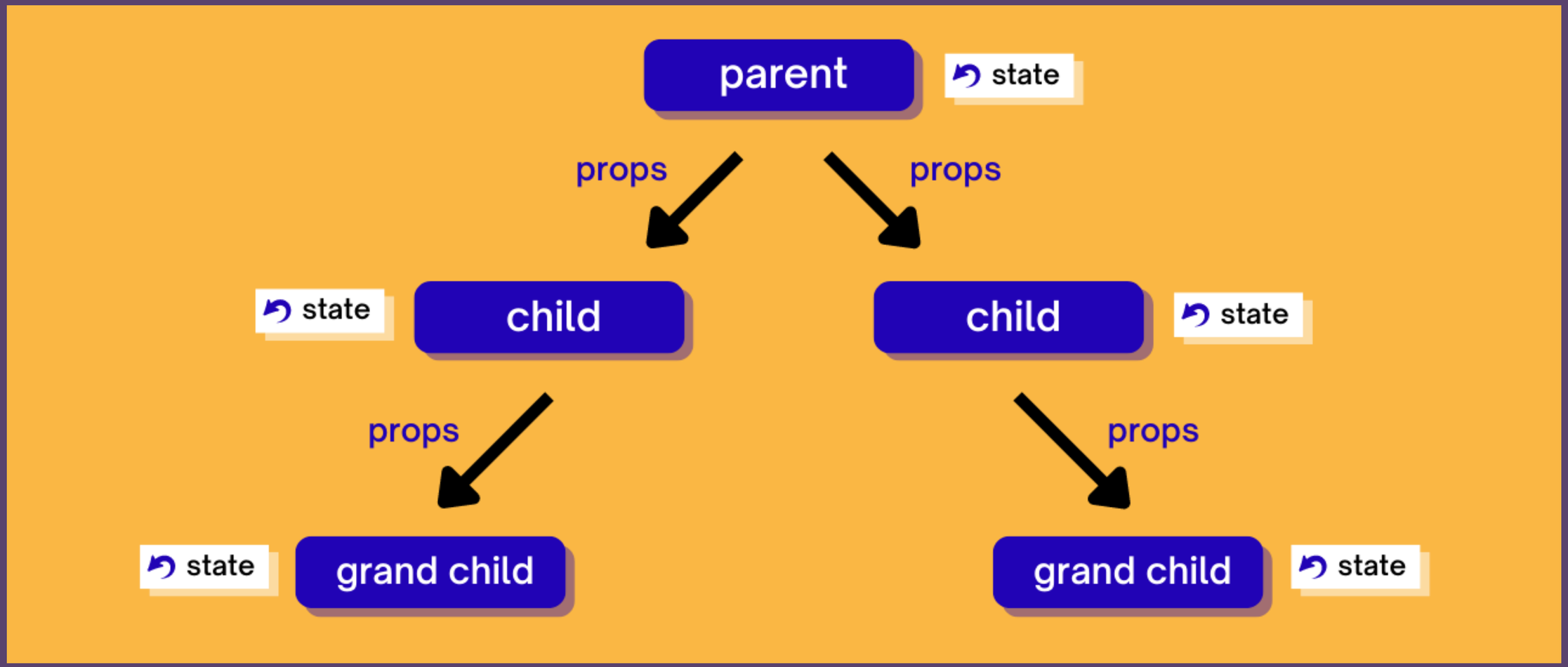
Passing Data via Props

Props in React

- Short for properties
- Mechanism for passing data.

- Read-only by default Usage
  - Pass data from parent to child component.
  - Makes components reusable.
- Defined as attributes in JSX. Key Points
  - Data flows one-way (downwards).
  - Props are immutable.
- Used for communication between components. Examples

```
<Header title="My App" />
```


parent
state
props
props
state
child
child
state
props
props
state
grand child
grand child
state

CSS Modules

1. Localized class names to avoid global conflicts.
2. Styles are scoped to individual components.
3. Helps in creating component-specific styles.
4. Automatically generates unique class names.
5. Promotes modular and maintainable CSS.
6. Can use alongside global CSS when needed.

Passing Children

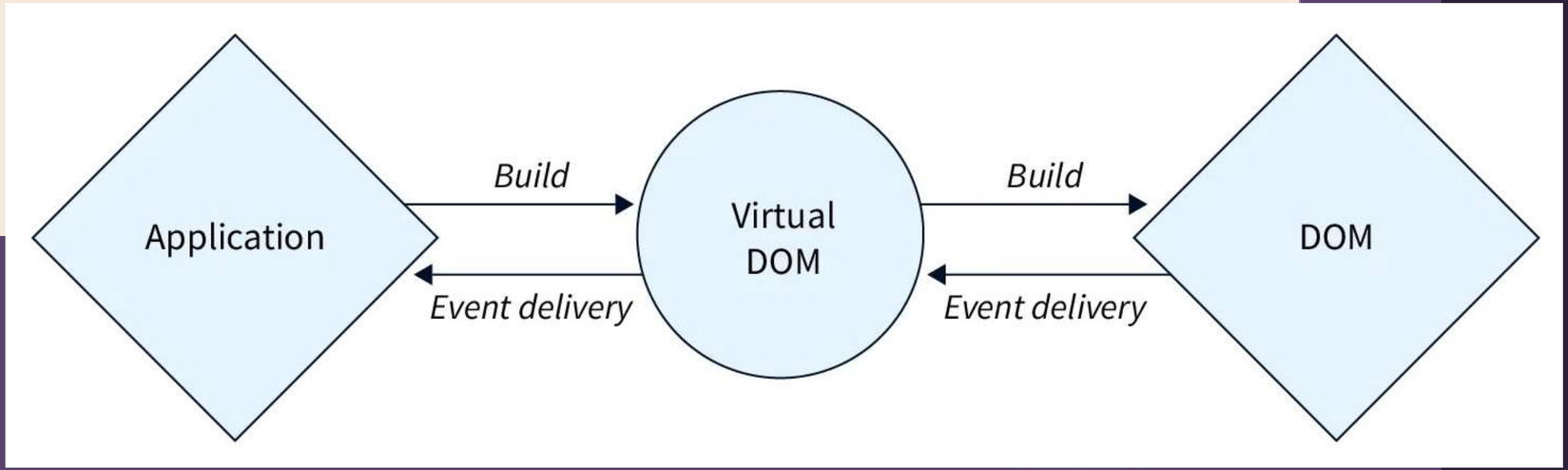
1. children is a special prop for passing elements into components.
2. Used for flexible and reusable component designs.
3. Common in layout or container components.
4. Accessed with props.children.
5. Can be any content: strings, numbers, JSX, or components.
6. Enhances component composability and reusability.

```
function Container(props) {
  return (
    <div className="container-style">
      {props.children}
    </div>
  );
}
```

```
<Container>
  <h1>Welcome to My App</h1>
  <p>This content is passed as children to the
  Container component.</p>
</Container>
```

Handling Events

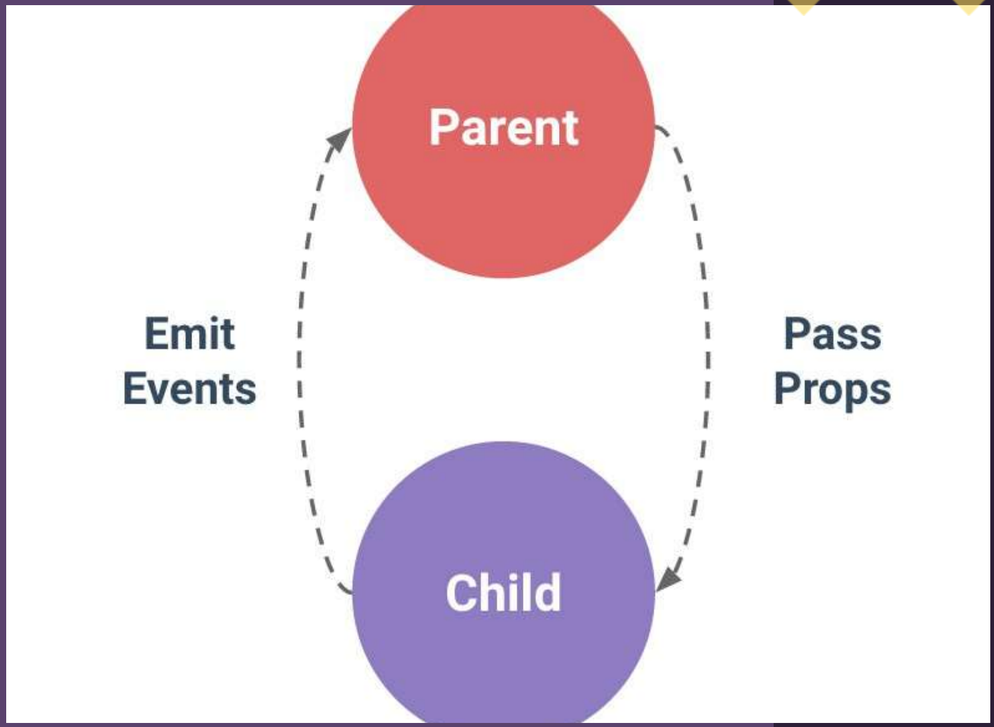
1. React events use camelCase, e.g., onClick.
2. Uses synthetic events, not direct browser events.
3. Event handlers can be functions or arrow functions.
4. Use onChange for controlled form inputs.
5. Avoid inline arrow functions in JSX for performance.



Passing Functions via Props

1. Pass dynamic behaviour between components.
2. Enables upward communication from child to parent.
3. Commonly used for event handling.
4. Parent defines a function, child invokes it.
5. Enhances component interactivity.
6. Example:

```
<Button onClick={handleClick} />
```



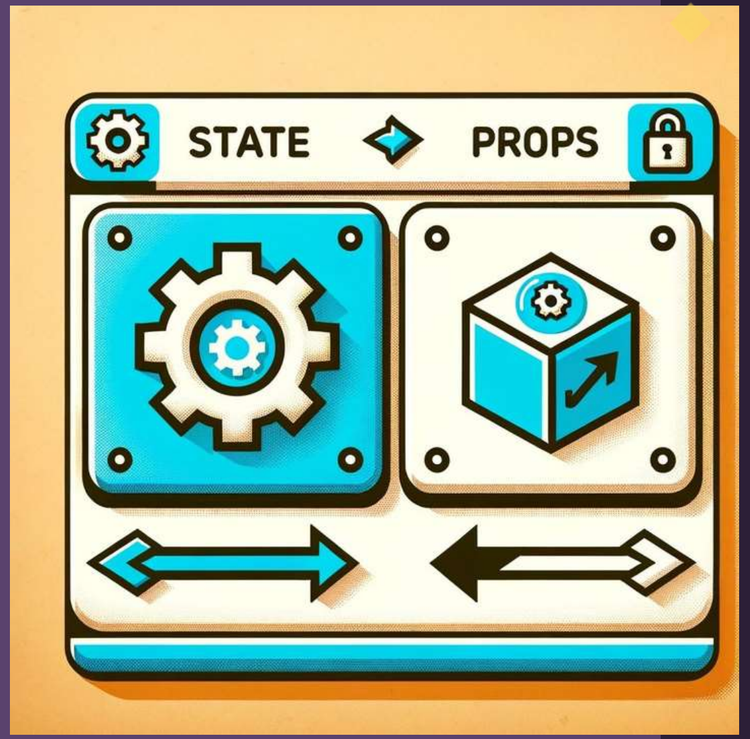
State vs Props

State:

- Local and mutable data within a component.
- Initialized within the component.
- Can change over time.
- Causes re-render when updated.
- Managed using useState in functional components.

Props:

- Passed into a component from its parent.
- Read-only (immutable) within the receiving component.
- Allow parent-to-child component communication.
- Changes in props can also cause a re-render.



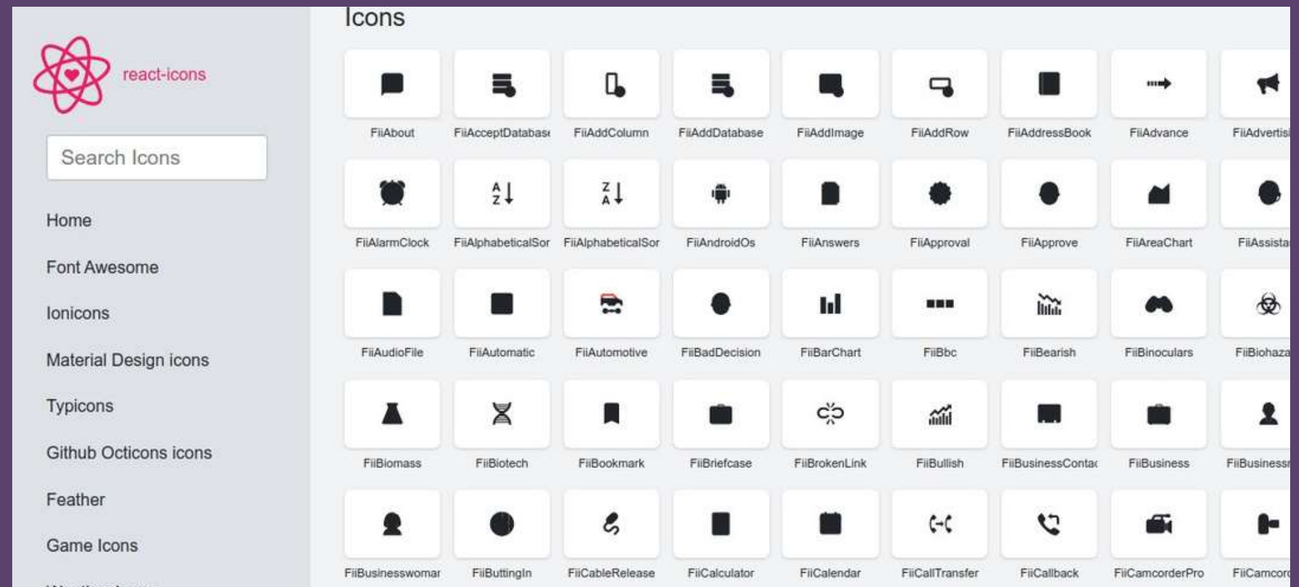
React-icon Library

1. You can use a lot of icons without managing them.
2. Install Package

npm install react-icons –save

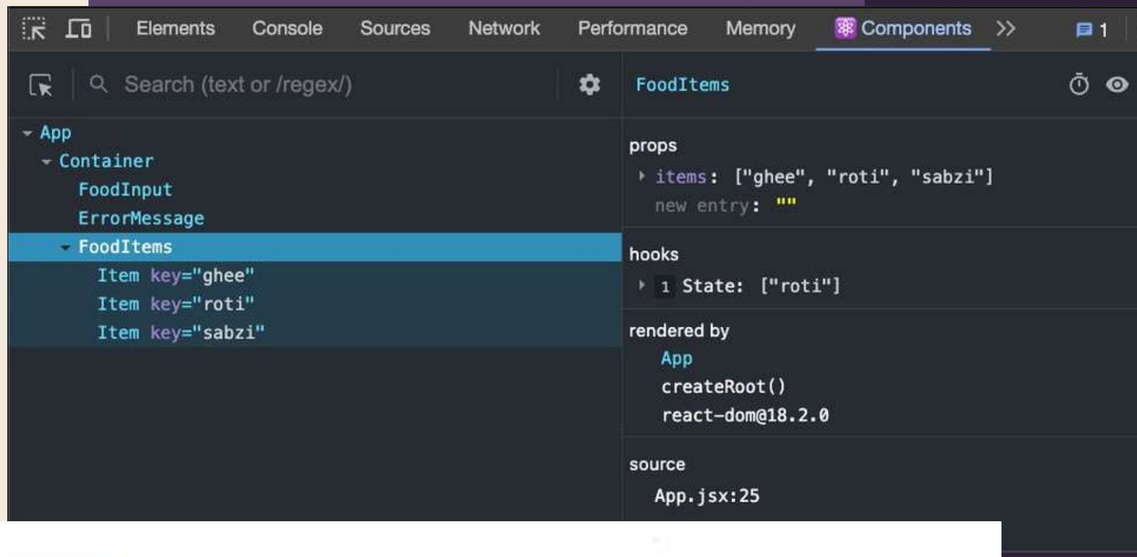
1. Use icon:

```
import {IconName} from "react-icons/fc";
```



Inspecting with React Dev Tools

1. Inspection: Allows inspection of React component hierarchies.
2. State & Props: View and edit the current state and props of components.
3. Performance: Analyze component re-renders and performance bottlenecks.
1. Navigation: Conveniently navigate through the entire component tree.
2. Filtering: Filter components by name o to locate them quickly.
1. Real-time Feedback: See live changes a modify state or props.



How React Works

Root Component:

- The App is the main or root component of a React application.
- It's the starting point of your React component tree.

Virtual DOM:

- React creates an in-memory structure called the virtual DOM.
- Different from the actual browser DOM.
- It's a lightweight representation where each node stands for a

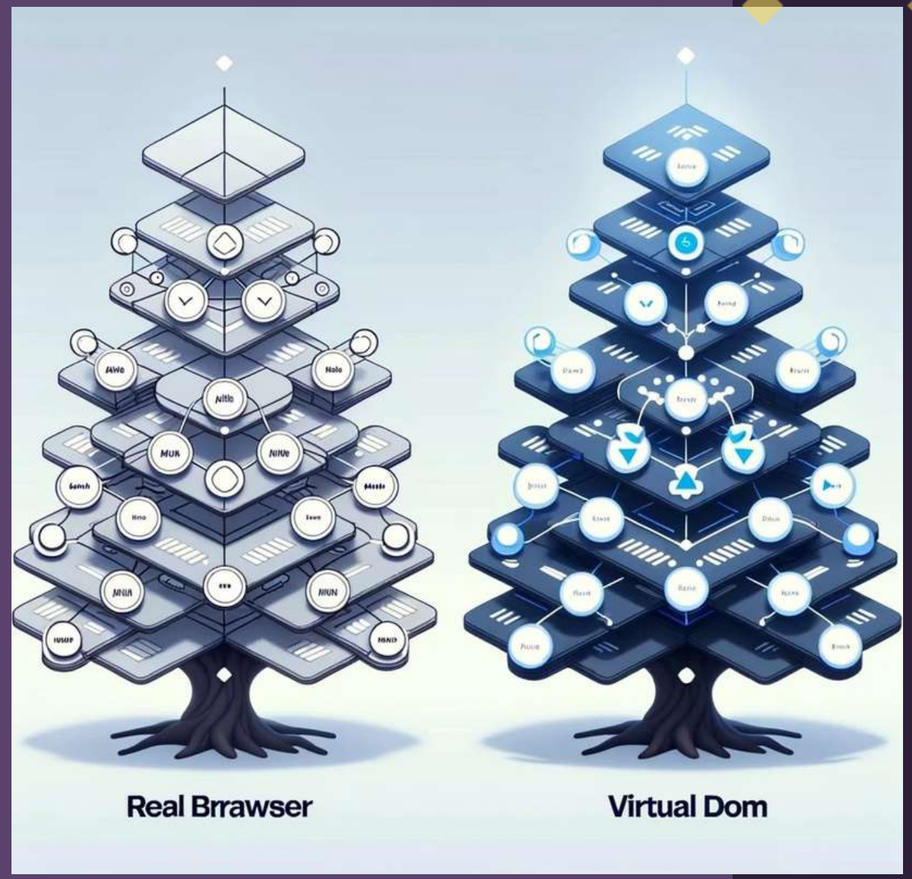
component and its attributes.

Reconciliation Process:

- When component data changes, React updates the virtual DOM's

state to mirror these changes.

- React then compares the current and previous versions of the virtual DOM.
- It identifies the specific nodes that need updating.
- Only these nodes are updated in the real browser DOM, making it efficient.



How React Works

React and ReactDOM:

- The actual updating of the browser's DOM isn't done by React itself.
- It's handled by a companion library called react-dom.

Root Element:

- The root div acts as a container for the React app.
- The script tag is where the React app starts executing.
- If you check main.tsx, the component tree is rendered inside this root element.

Strict Mode Component:

- It's a special component in React.
- Doesn't have a visual representation.
- Its purpose is to spot potential issues in your React app.

Platform Independence:

- React's design allows it to be platform-agnostic.
- While react-dom helps build web UIs using React, ReactNative can be used to craft mobile app UIs.

React Vs Angular vs Vue

React, Angular, and Vue:

- React is a library, while Angular and Vue.js are frameworks.
- React focuses on UI; Angular and Vue.js offer comprehensive tools for full app development.

Library vs. Framework:

- A library offers specific functionality.
- A framework provides a set of tools and guidelines.
- In simpler terms: React is a tool; Angular and Vue.js are toolsets.

React's Specialty:

- React's main role is crafting dynamic, interactive UIs.
- It doesn't handle routing, HTTP calls, state management, and more.

React's Flexibility:

- React doesn't dictate tool choices for other app aspects.
- Developers pick what fits their project best.

About Angular and Vue.js:

- Angular, developed by Google, provides a robust framework with a steep learning curve.
- Vue.js is known for its simplicity and ease of integration, making it beginner-friendly.

initial value:
Pass the initial value to the useRef hook

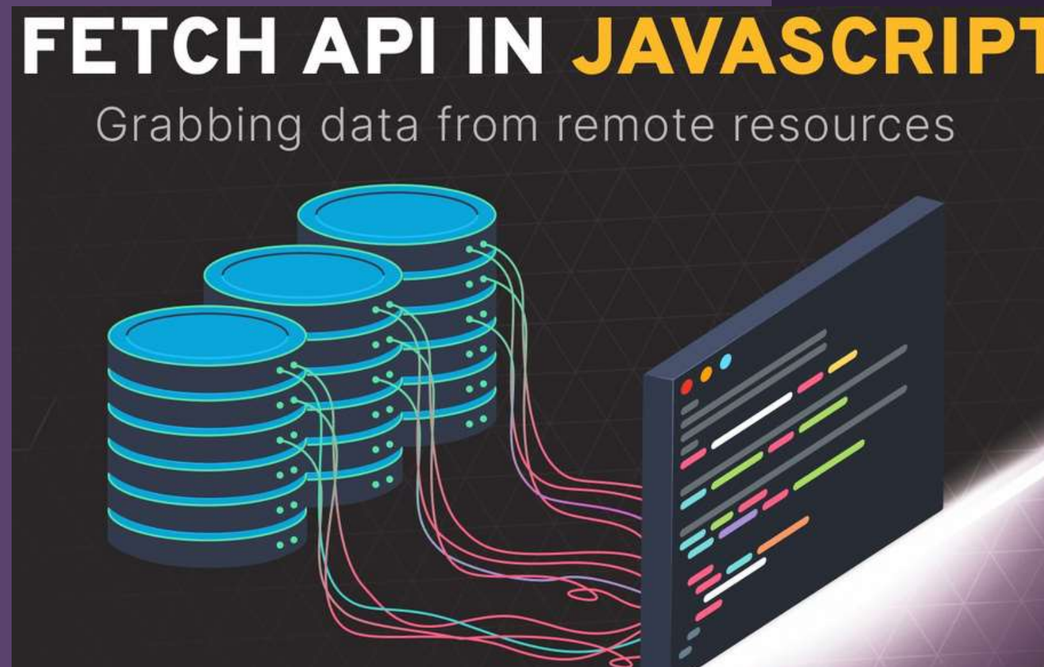
```
const refObject = useRef( initialValue );
```

current

ref:
React returns an object with a current property

Data fetching using Fetch

1. fetch: Modern JavaScript API for network requests.
2. Promise-Based: Returns a Promise with a Response object.
3. Usage: Default is GET. For POST use method: 'POST'
4. Response: Use .then() and response.json() for JSON data.
5. Errors: Doesn't reject on HTTP errors. Check response.ok.
6. Headers: Managed using the Headers API.



Thank You !!!!!