1. **What is React JS?**
    - Frontend JavaScript framework
    - React is a **JavaScript library** for building user interfaces.
    - React is used to build **single-page applications (SPA)**
    - React allows us to **create reusable UI components**.(**Components** is a self contained section of code that functions as a reusable building blocks)
2. **ES6**
    - React ES6 refers to the use of ECMAScript 6 (ES6) features and syntax within React applications.
        - **Classes** - ES6 introduced the class syntax for defining JavaScript classes, which is used in React for creating class components.
        - **Arrow Functions** - ES6 arrow functions provide a concise syntax for defining functions, commonly used for React components and event handlers.
        - **Variables (let, const, var) -** let is used for variables that can be reassigned, while const is used for variables that are meant to be constant.
        - **Array Methods like .map()**
        - **Destructuring -** Destructuring is a handy feature in JavaScript that lets you extract values from objects or arrays and assign them to variables in a simpler and more readable way. (i.e.,Destructuring is a handy feature in JavaScript that lets you extract values from objects or arrays and assign them to variables in a simpler and more readable way.)
        - **Modules -** Modules in ES6 are like separate compartments where you can neatly organize your code. They provide a standardized way to share functionality between different parts of your application.
        - **Ternary Operator -** The ternary operator is a concise way to write conditional statements in JavaScript. It's often used when you have a simple if-else statement and want to assign a value based on a condition.
        - Example:
          const message = age >= 18 ? 'You are an adult' : 'You are a minor';
        - **Spread Operator -** The spread operator (...) is used to expand an iterable object (like an array) into individual elements. It's commonly used to make copies of arrays, concatenate arrays, or pass multiple arguments to a function.

3. React Render HTML

- React renders HTML to the web page by using a function called createRoot() and its method render().
- **Example:**

```
import React from 'react';
// A simple React component that renders HTML
const MyComponent = () => {
  return (
    <div>
      <h1>Hello, React!</h1>
      <p>This is a paragraph.</p>
    </div>
  );
};
export default MyComponent;
```

4. Advantages of React JS?

- Reusable components
- Open Source
- Fast and efficient
- Work in Browser
- Large community

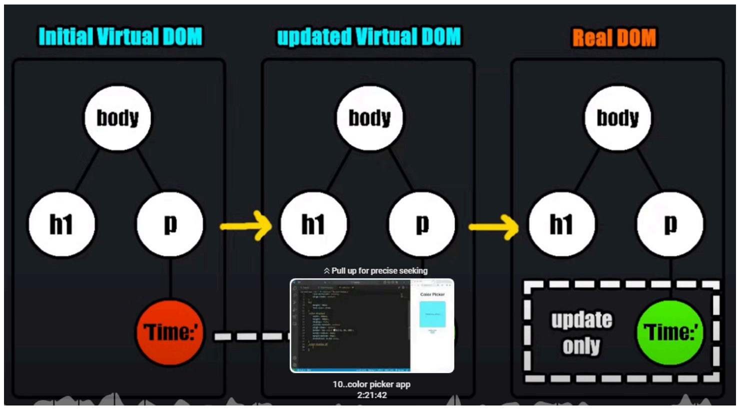
5. Notes:

- **One component will return one child only.**
- Instead of using <div></div> multiple times, <fragment></fragment> tag can be used. Inside the fragment tag we can give multiple div's **OR** we can use <></> empty tags.
- <Header/> this is called standalone tag

6. How does React Work?

- React **creates a VIRTUAL DOM in memory.**
- Instead of manipulating the browser's DOM directly, React creates a virtual DOM in memory, where it does all the necessary manipulating, before making the changes in the browser DOM.

- **React only changes what needs to be changed!**
- **Ref Image:**



7. How to run a react program in vite.

- npm create vite@latest
- Type Project Name
- Choose React & JavaScript
- These 3 lines one by one
    - cd card-component
    - npm install
    - npm run dev

How to run React Program without vite

- npx create-react-app my-react-app
- cd my-react-app
- npm start

Explanation

- Main.jsx file is the entry point of react
- Html+Js = JSX
- App.jsx - is a function based components

8. JSX (JavaScript XML)

- Javascript **syntax extension** that allows us to write HTML & Javascript in a single file

- JSX allows you to write HTML elements in Javascript & place them in the DOM without any **createElement() or appendChild()** methods.

- JSX **Converts** HTML Tags into React Elements.

- **Babel** is used in React to convert JSX to actual Javascript (It works in backend)

- **https://babeljs.io/ converts JSX to JS**

- In JSX, Expressions can be written inside curly braces **(Example : <p>25+45={25+45}</p> the output will be displayed : 25+45=70**

9. **React components**

   React components are JavaScript functions or classes that return JSX (JavaScript XML) to describe the UI. They can represent anything from a simple button or text input to complex UI elements like forms, modals, or entire pages.

   - **Class component** - Header is a class component that extends React.Component. It receives a prop (title) and renders an <h1> element with the title.
   - **Function Component** - Button is a functional component that accepts props (onClick and label) and returns a <button> element with an onClick event handler and a label.

10. **Component Constructor**
    - Think of the constructor as a special setup area for a component. It's like preparing a recipe before you start cooking. In React, the constructor is where you set up things like what the component's starting state is and what functions it needs.

11. **React Forms**
    - In React, forms are created using standard HTML form elements like <input>, <textarea>, and <select>. However, React adds a layer of interactivity and control by managing the form state and handling form submission using JavaScript.

12. **React Router**
    - React Router provides a way to manage navigation and URL changes within your React application without causing a page reload. It allows you to define routes that map URLs to specific components, enabling a more dynamic and interactive user experience.

13. **React Memo**
    - Using memo will cause React to skip rendering a component if its props have not changed.

14. **React SASS Styling**
    - Sass is a CSS pre-processor.

15. **Conditional Rendering**
    - It works based on the condition declared
    - **Example**

```
import React from 'react';
function Greeting(props) {
  const currentTime = new Date().getHours();
```

```
 const greetingMessage = currentTime < 12 ? 'Good morning!' : 'Good afternoon!';
 return (
  <div>
   <h1>Greeting</h1>
   <p>{greetingMessage}</p>
  </div>
 );
}
export default Greeting;
```

16. JSX with List

- **JavaScript map(**) array method is generally the preferred method for lists.
- We can list the items using Map Methods
- **Example**

```
import React from 'react';
function FruitList() {
 const fruits = ['Apple', 'Banana', 'Orange', 'Mango'];
 return (
  <div>
   <h1>Fruit List</h1>
   <ul>
    {fruits.map((fruit, index) => (
     <li key={index}>{fruit}</li>
    ))}
   </ul>
  </div>
 );
}
export default FruitList;
```

Output :

```
Fruit List
- Apple
- Banana
- Orange
- Mango
```

17. **What is rendering?**

Rendering in React JS can be summarized as the process of converting React components into DOM (Document Object Model) elements and displaying them in the browser.

18. **Props**

- **Properties** are known as Props
- Props are read-only and are passed down from parent components to child components.
- You can pass as many arguments as needed to customize the behavior or appearance of the component.

Example:

```
import React from 'react';
function Greeting(props) {
  return <h1>Hello, {props.name}!</h1>;
}
function App() {
  return (
    <div>
      <Greeting name="Alice" />
      <Greeting name="Bob" />
    </div>
  );
}
export default App;
```

Output:

```
Hello, Alice!
Hello, Bob!
```

19. **Default Props**

- Default props in React allow you to specify default values for props that are used by a component when the prop is not explicitly provided. This ensures that your component behaves predictably even if certain props are not passed to it.
- **Example :**

```
import React from 'react';
class Greeting extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}!</h1>;
  }
}

Greeting.defaultProps = {
  name: 'Guest' // Default value for the 'name' prop
};
export default Greeting;
```

Output:

```
Hello, Guest!
```

20. Child Component

- A child component refers to a component that is nested within another component, often referred to as the parent component. When a parent component renders a child component, it can pass data to the child component using props, allowing the child component to receive and utilize that data.
- **Example:**

```
import React from 'react';

// Child component
function Child(props) {
  return <p>Hello, {props.name}!</p>;
}

// Parent component
function Parent() {
  return (
    <div>
      <h1>Parent Component</h1>
      <Child name="Alice" />
    </div>
  );
}
export default Parent;
```

Explanation:

We define a functional component Child which accepts a name prop and renders a greeting message using that prop.

We define another functional component Parent which renders a heading <h1> and then renders the Child component within it, passing the name prop with the value "Alice".

The name prop is passed from the Parent component to the Child

21. How to pass an array as props?

- To pass an array as props in React, you can simply pass the array as a prop value when rendering the component.
- **Example:**

```
import React from 'react';
function MyComponent(props) {
  return (
    <div>
      <h1>Items:</h1>
      <ul>
        {props.items.map((item, index) => (
          <li key={index}>{item}</li>
        ))}
      </ul>
    </div>
  );
}
function App() {
  const items = ['Apple', 'Banana', 'Orange'];

  return <MyComponent items={items} />;
}
export default App;
```

22. **oneOf & oneOfType**

- In PropTypes, **oneOf** is a validator that ensures that the value of a prop is one of a specified set of values.
- **oneOfType** is a PropTypes validator in React that checks whether a prop matches at least one of the specified PropTypes.
- **oneOfType** is particularly useful when you have a prop that can accept values of different types, and you want to validate that at least one of those types is passed correctly.

23. **React Events**

- **Click Event:** Triggered when a user clicks on a DOM element.
- **Change Event:** Fired when the value of an input element changes, like in text input fields, checkboxes, or select dropdowns.
- **Mouse Events:** Including events like mouseover, mouseout, mouseenter, and mouseleave, which are triggered when the mouse interacts with DOM elements.

24. **Hooks?**

- Hooks enable functional components to have local state, perform side effects, and access other React features without needing to convert them into class components.
  - **useState**
  - **useEffect**
  - useContext
  - useRef
  - useReducer
  - useCallback
  - useMemo

25. **useState**

- **"useState" hook is a feature that allows developers to add state variables to functional components.** This means you can create and manage state within a component, giving it the ability to store and update data dynamically.
- In simpler terms, think of "useState" as a way for React components to keep track of information that might change over time, such as user input, component visibility, or any other dynamic data needed for your application. It's like having a memory within each

component, allowing it to remember and respond to changes, making your user interfaces more interactive and responsive.

- **Example:**

```
import React, { useState } from 'react';
function Counter() {
 // Define a state variable called "count" and a function to update it, "setCount"
 const [count, setCount] = useState(0);
 // Define a function to increment the count when a button is clicked
 const increment = () => {
  setCount(count + 1); // Update the count state
 };
 return (
  <div>
   <p>Count: {count}</p>
   <button onClick={increment}>Increment</button>
  </div>
 );
}
export default Counter;
```

26. useEffect

- useEffect is a Hook that allows you to perform side effects in function components.
- useEffect is commonly used for actions **like data fetching, subscriptions, or manually changing the DOM in React components.**
- **Example:**

```
import React, { useState, useEffect } from 'react';
function MyComponent() {
 const [data, setData] = useState(null);
 useEffect(() => {
  // This function will be called after the component renders.
  // It's the equivalent of componentDidMount and componentDidUpdate.
  // Data fetching example
  fetch('https://api.example.com/data')
   .then(response => response.json())
   .then(data => setData(data))
   .catch(error => console.error('Error fetching data:', error));
  // Clean-up function (equivalent to componentWillUnmount)
  return () => {
   // Perform any clean-up actions here, such as canceling timers or subscriptions.
  };
```

27. **useContext**
    - **useContext** is used for accessing shared values across components without manually passing props through each level.
        - **Create a context**
          ```
          Import {createContext} from 'react';
          Export const cartContext = createContext();
          ```
        - **Provide the context**
          ```
          <cartContext.Provider value = {value}>
          <child/>
          <child/>
          <child/>
          </cartContext.Provider>
          ```
        - **Consume the context**
          ```
          Import {cartContext} from "./cartContext";
          Const {cart} = useContext(cartContext);
          ```
28. **Prop drilling** in React refers to passing props through multiple layers of nested components. While it's a common pattern, it can lead to code that's harder to maintain as the application grows.
29. **useRef**
    - useRef is a hook in React that provides a way to access and interact with DOM elements or other React components directly.
30. **useReducer**
    - useReducer is particularly useful for managing complex state logic, such as forms, data fetching, or any state that involves multiple sub-values or transitions. It helps keep the state management code organized and easier to reason about, especially in larger applications.
31. **useCallback**
    - useCallback is like a memory aid for functions in React. It helps React remember functions so that they don't change unnecessarily.

32. useMemo

- The React useMemo Hook returns a memoized value.