C20452282

Lab4

Theary questions

Q1 Props are used to pass data from a parent component to a child component. They are immutable, meaning their values cannot be changed within the component. Props are essential for communication between components, allowing you to configure child components with data from their parent.

Example:

// ParentComponent.js

import React from 'react';

import ChildComponent from './ChildComponent';

const ParentComponent = () => {

const message = 'Hello from Parent!';

return <ChildComponent greeting={message} />;

};

// ChildComponent.js

import React from 'react';

const ChildComponent = (props) => {

return <p>{props.greeting}</p>;

};

export default ChildComponent;

State is used to manage the internal data of a component. Unlike props, state can be changed, and when it changes, React re-renders the component to reflect the updated state

import React, { useState } from 'react';

const CounterComponent = () => {

const [count, setCount] = useState(0);

const increment = () => {

setCount(count + 1);

};

const decrement = () => {

setCount(count - 1);

};

return (

<div>

<p>Count: {count}</p>

<button onClick={increment}>Increment</button>

<button onClick={decrement}>Decrement</button>

</div>

);

};

export default CounterComponent;

Q3

In functional programming, a functor is an object or data structure that implements map function, allowing you to apply a function to the values inside the functor while preserving the functor's structure. Functors provide a way to perform transformations on values in a context (such as an array, a container, or an option type) without unwrapping or modifying the structure of that context.

// Functor: Array

const myArray = [1, 2, 3, 4, 5];

// Mapping function

const double = (value) => value \* 2;

// Functor implementation

const newArray = myArray.map(double);

console.log(newArray); // [2, 4, 6, 8, 10]

Q3

Call backs

Advantages: callbacks are straightforward and easy to understand, especially for simple asynchronous operations.

Disadvantages: Asynchronous operations often lead to nested callback functions, resulting in what is commonly known as "callback hell" or the "pyramid of doom." This can make the code hard to read, maintain, and debug.

Promesis

Advantages : promises allow you to chain multiple asynchronous operations using .then() This chaining results in more readable and maintainable code, often referred to as the "Promise Chain."

Disadvantages: Promises have an unsettled state, which means they can be in a pending state, resolved, or rejected. This introduces complexity when dealing with error handling and may require additional handling for race conditions.

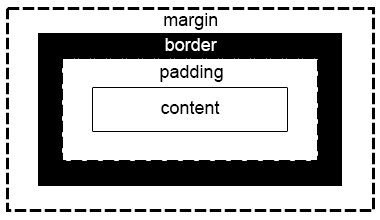
Streams

Advantage: Streams allow for backpressure, meaning the consumer can control the rate at which data is consumed. This helps manage resources efficiently, especially when dealing with large datasets or slow consumers.

Disadvantages: Streams might have a steeper learning curve compared to callbacks and promises, especially for beginners. Understanding concepts like piping, transforming, and handling errors in a stream can take some time to master.

Q4

The css box is a fundamental concept in web development that describes how elements on a webpage are structured and spaced. It comprises the content area, padding, border, and margin of an element. Understanding the box model is crucial for styling and layout purposes.



<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<style>

.box {

width: 200px;

height: 100px;

padding: 20px;

border: 5px solid #3498db;

margin: 30px;

}

.box2 {

width: 150px;

height: 75px;

padding: 10px;

border: 2px solid #e74c3c;

margin: 15px;

}

</style>

<title>CSS Box Model Example</title>

</head>

<body>

<div class="box">

<p>This is a box with content, padding, border, and margin.</p>

</div>

<div class="box2">

<p>Another box with different dimensions and styles.</p>

</div>

</body>

</html>

Q5

The browser follows a series of steps to load and bootstrap a rich web application. It starts with fetching and parsing the initial HTML, loading external resources, executing scripts, and finally, initializing the application. As the user interacts with the application, additional resources may be loaded dynamically, and the application responds accordingly.