NAME:-OMATIGA JOSHUA IKO-OJO

MATRIC NO:-23/2000

DEPARTMENT :- 200 LEVEL INFORMATION TECHNOLOGY (GROUP C)

**BU-COS 209**

ASSIGNMENT 1

**CLASSES**

In React, class components are defined by extending React.Component and include methods like constructor , render, and lifecycle methods (componentDidMount, componentWillUnmount). They manage their own state using this.state and update it with this.setState. An example is a Counter component that increments a count when a button is clicked. While class components are still useful for understanding React, functional components with hooks are now the preferred approach.

import React, { Component } from 'react';

class Counter extends Component {

constructor(props) {

super(props);

this.state = { count: 0 };

}

increment = () => {

this.setState(prevState => ({ count: prevState.count + 1 }));

};

render() {

return (

<div>

<h1>Counter: {this.state.count}</h1>

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

</div>

);

}

}

export default Counter;

**ARROW FUNCTIONS**

Arrow functions in React simplify function definitions and automatically bind this to the surrounding context, making them especially useful in class components for event handlers.

Key Points:

- Lexical this Binding : Arrow functions inherit this from their parent scope.

- Concise Syntax: They provide a shorter way to write functions.

Example in Class Component:

javascript

handleClick = () => {

this.setState(prevState => ({ count: prevState.count + 1 }));

};

```

Example in Functional Component:

javascript

const handleClick = () => {

setCount(prevCount => prevCount + 1);

};

Overall, arrow functions enhance code readability and reduce the need for manual binding in class components.

**VARIABLES**

In React, there are three main types of variables used to manage data:

1. Local Variables: Defined within a component for temporary storage; they do not trigger re-renders when changed.

javascript

let message = "Hello, World!";

2. State Variables: Managed with useState (or this.state in class components); changes to state variables trigger re-renders.

javascript

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

3. Props: Used to pass data from parent to child components; props are immutable in the child.

javascript

const Greeting = ({ name }) => <h1>Hello, {name}!</h1>;

These variable types are essential for handling data and rendering dynamic content in React applications.

**ARRAY METHODS**

The .map() method in JavaScript creates a new array by applying a function to each element of the original array, leaving the original array unchanged.

Key Points:

- Syntax:

javascript

const newArray = array.map((element) => {

// return transformed element

});

- Example: Doubling numbers in an array:

javascript

const numbers = [1, 2, 3];

const doubled = numbers.map(num => num \* 2); // [2, 4, 6]

- Use in React: Commonly used to render lists of components:

javascript

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

The .map() method is essential for transforming data and rendering dynamic content efficiently in JavaScript and React.

**DESTRUCTURING**

Destructuring is a JavaScript feature that simplifies extracting values from arrays and properties from objects. In React, it's commonly used for:

Key Points:

1. Array Destructuring: Allows you to assign elements to variables.

javascript

const [first, second] = ['red', 'green']; // first: 'red', second: 'green

2. Object Destructuring: Simplifies access to object properties, particularly useful for props.

javascript

const { name, age } = { name: 'Alice', age: 30 }; // name: 'Alice', age: 30

3. Destructuring in Functional Components: Directly destructures props in the function parameters.

javascript

const Greeting = ({ name, age }) => <h1>Hello, {name}!</h1>;

4. Destructuring in Class Components: Used to extract state properties in the `render` method.

javascript

const { name, age } = this.state;

5. Nested Destructuring : Allows extraction from nested objects.

javascript

const { profile: { name } } = user;

Destructuring improves code readability and reduces redundancy in React applications.

**MODULES**

In React, modules help organize code by encapsulating functionality into separate files, promoting reusability and maintainability.

Key Points:

1. Modules: Use `import` and `export` syntax to manage code.

2. Exporting:

- Named Exports: Allow multiple items to be exported.

javascript

export const add = (a, b) => a + b;

- Default Exports: Allow a single item to be exported.

javascript

export default Greeting;

3. Importing:

- Named Exports:

javascript

import { add } from './mathUtils';

- Default Exports:

javascript

import Greeting from './Greeting';

4.

Organization: Commonly structured in directories for components and utilities.

5. Custom Hooks: Modules can encapsulate reusable logic in hooks.

Using modules enhances clarity and helps manage complex React applications efficiently.

**TERNARY OPERATOR**

The ternary operator is a concise conditional expression in JavaScript, useful in React for rendering elements based on conditions.

Key Points:

1. Syntax:

javascript

condition ? valueIfTrue : valueIfFalse

2. Usage in JSX:

- Basic Rendering:

javascript

{isLoggedIn ? <h1>Welcome back!</h1> : <h1>Please log in.</h1>}

3. Rendering Multiple Elements:

javascript

{user ? <h1>Hello, {user.name}!</h1> : <h1>No user logged in.</h1>}

4. Conditional Styling:

javascript

<button className={isActive ? 'active' : 'inactive'}>{isActive ? 'Active' : 'Inactive'}</button>

The ternary operator enhances code readability and enables cleaner conditional rendering in React applications.

**SPREAD OPERATOR**

The spread operator (`...`) in React is a JavaScript feature that allows you to expand elements of an iterable (like arrays or objects). Here are its main uses:

1. Spreading Props: Pass all properties of an object to a component easily.

jsx

<UserProfile {...user} />

2. Updating State: Update state immutably by spreading existing state properties.

jsx

setUser(prevUser => ({ ...prevUser, age: 26 }));

3. Combining Arrays: Merge arrays effortlessly.

jsx

const allFruits = [...fruits, ...moreFruits];

4. Cloning: Create shallow copies of objects and arrays.

jsx

const clone = { ...original };

Overall, the spread operator improves code readability and helps manage data structures in an immutable way.