METHOD AS PROPS

VISHNU C
287616

INTRODUCTION

What Are Props in React?

- Props allow passing data from parent to child components.
- In TypeScript, props are strongly typed for better type safety and code quality.

WHY PASS METHODS AS PROPS?

The Need for Methods as Props

- Allows child components to trigger actions in parent components.
- Supports callback functionality and event handling
- Encourages component reusability and separation of concerns

DEFINING METHOD AS PROPS

Basic Structure

- In React, functions are passed to child components as props.
- TypeScript ensures that these functions follow a specific type signature.

```
interface ButtonProps {
  onClick: () => void;
}
```

ADVANTAGES

- **Separation of Concerns:** Logic is kept in the parent, while rendering happens in the child.
- **Reusability:** Child components can trigger various functions without changing the component.
- **Testability:** Easier to test methods passed as props.
- **Type Safety:** TypeScript ensures proper function signatures.

PROPS TYPES

I. Primitive Types

```tsx
interface MyComponentProps {
  name: string;
  age: number;
  isStudent: boolean;
}

const MyComponent: React.FC<MyComponentProps> = ({ name, age, isStudent }) => {
  return (
    <div>
      <p>Name: {name}</p>
      <p>Age: {age}</p>
      <p>Student: {isStudent ? "Yes" : "No"}</p>
    </div>
  );
};
```

II. Optional Props

```tsx
interface MyComponentProps {
  name: string;
  age?: number; // Optional prop
}

const MyComponent: React.FC<MyComponentProps> = ({ name, age = 18 }) => {
  return (
    <div>
      <p>Name: {name}</p>
      <p>Age: {age}</p>
    </div>
  );
};
```

III. Array Types

```tsx
interface MyComponentProps {
  hobbies: string[];
}

const MyComponent: React.FC<MyComponentProps> = ({ hobbies }) => {
  return (
    <div>
      <h3>Hobbies:</h3>
      <ul>
        {hobbies.map((hobby, index) => (
          <li key={index}>{hobby}</li>
        ))}
      </ul>
    </div>
  );
};
```

IV. Object Types

```tsx
interface Address {
  street: string;
  city: string;
}

interface MyComponentProps {
  name: string;
  address: Address;
}

const MyComponent: React.FC<MyComponentProps> = ({ name, address }) => {
  return (
    <div>
      <p>Name: {name}</p>
      <p>Address: {address.street}, {address.city}</p>
    </div>
  );
};
```

V. Function Types

```tsx
interface MyComponentProps {
  onClick: (message: string) => void;
}

const MyComponent: React.FC<MyComponentProps> = ({ onClick }) => {
  return <button onClick={() => onClick("Button clicked!")}>Click me</button>;
};
```

VI. Union Types

```tsx
interface MyComponentProps {
  id: string | number;
}

const MyComponent: React.FC<MyComponentProps> = ({ id }) => {
  return <p>ID: {id}</p>;
};
```

VII. Enum Types

```tsx
enum ButtonVariant {
  Primary = "primary",
  Secondary = "secondary",
}

interface MyComponentProps {
  variant: ButtonVariant;
}

const MyComponent: React.FC<MyComponentProps> = ({ variant }) => {
  return <button className={variant}>Click me</button>;
};
```

VIII. Default Props

```tsx
interface MyComponentProps {
  name?: string;
}

const MyComponent: React.FC<MyComponentProps> = ({ name = "Guest" }) => {
  return <p>Hello, {name}!</p>;
};
```

IX. Generic Props

```tsx
interface MyComponentProps<T> {
  items: T[];
}

const MyComponent = <T extends unknown>({ items }: MyComponentProps<T>) => {
  return (
    <ul>
      {items.map((item, index) => (
        <li key={index}>{item}</li>
      ))}
    </ul>
  );
};
```

THANK YOU