Form validation in React ensures that users enter valid and complete data before submitting a form. You can handle it manually using state and logic, or use libraries like **React Hook Form** for a cleaner, more scalable approach.

✅ **Manual Validation (Using useState)**

Here's a simple example:

```
import { useState } from 'react';


function SimpleForm() {
  const [email, setEmail] = useState('');
  const [error, setError] = useState('');


  const handleSubmit = (e) => {
    e.preventDefault();
    if (!email.includes('@')) {
      setError('Invalid email address');
    } else {
      setError('');
      alert('Form submitted!');
    }
  };


  return (
    <form onSubmit={handleSubmit}>
      <input
        type="text"
        value={email}
        onChange={(e) => setEmail(e.target.value)}
        placeholder="Enter email"
      />
      {error && <p style={{ color: 'red' }}>{error}</p>}
      <button type="submit">Submit</button>
    </form>
```

```
  );
}
```

Using react-hook-form (Recommended for larger forms)

This library simplifies validation and reduces boilerplate:

```
npm install react-hook-form
import { useForm } from 'react-hook-form';

function HookForm() {
  const { register, handleSubmit, formState: { errors } } = useForm();

  const onSubmit = (data) => {
    console.log(data);
  };

  return (
    <form onSubmit={handleSubmit(onSubmit)}>
      <input
        {...register('email', { required: 'Email is required', pattern: /^\S+@\S+$/i })}
        placeholder="Email"
      />
      {errors.email && <p>{errors.email.message}</p>}
      <button type="submit">Submit</button>
    </form>
  );
}
```

Bonus Tips

- Use onBlur for real-time validation.
- Show error messages clearly.
- Disable the submit button until the form is valid.

In React, **passing a function as a prop** lets a parent component give control or behavior to a child component—like saying, “Hey, when this happens, run *this*.”

🔧 **How to Do It**

1. Define the function in the parent

```
function App() {
  const handleClick = () => {
    alert('Button clicked!');
  };

  return <Child onClick={handleClick} />;
}
```

2. Use it in the child

```
function Child({ onClick }) {
  return <button onClick={onClick}>Click Me</button>;
}
```

Important: Pass the function reference, not the result.
✅ onClick={handleClick}
❌ onClick={handleClick()} ← this would call it immediately on render.

Passing Arguments

If you want to pass arguments from the child:

```
function Child({ onClick }) {
  return <button onClick={() => onClick('Saumyajit')}>Greet</button>;
}
```

🧩 **Why It’s Powerful**

- Enables **child-to-parent communication**
- Keeps logic centralized in the parent
- Makes components **reusable** and **decoupled**