React JS Course in Bangla – React JS Bangla Tutorial

react-bangla.vercel.app/learn-react-quick-start-guide

> React শেখার শুরু: এক লেখায় ৮০% কনসেপ্ট ক্লিয়ার!

এই একটি পেজ যদি তুমি মন দিয়ে পড়ো আর বুঝে প্রাকটিস করো, বিশ্বাস করো, **React-এর প্রায় ৮০% জরুরি কনসেপ্ট তোমার হাতের মুঠোয় চলে আসবে**, যা তোমার প্রতিদিনের কোডিং জীবনে কাজে লাগবে।

এই পাঠ থেকে কি কি শিখবে

- কীভাবে কম্পোনেন্ট বানিয়ে একটার ভেতরে আরেকটা বসাতে হয়।
- কীভাবে মার্কআপ (HTML-এর মতো কোড) এ স্টাইল যোগ করতে হয়।
- কীভাবে স্ক্রিনে ডেটা দেখাতে হয়।
- কীভাবে শর্ত ও তালিকা অনুযায়ী UI Element রেন্ডার করতে হয়।
- কীভাবে ইউজারের কাজে (Events) সাড়া দিয়ে স্ক্রিন আপডেট করতে হয়।
- কীভাবে কম্পোনেন্টগুলোর মধ্যে ডেটা শেয়ার করতে হয়।

কম্পোনেন্ট বানানো ও একটার ভেতরে আরেকটা রাখা

React অ্যাপ তৈরি হয় অনেকগুলো ছোট ছোট **কম্পোনেন্ট (Component)** দিয়ে। একটা কম্পোনেন্ট হলো তোমার ইউজার ইন্টারফেসের (UI) একটা টুকরো, যার নিজস্ব লজিক এবং চেহারা আছে। এটা একটা ছোট্ট বাটন থেকে শুরু করে পুরো একটা পেজও হতে পারে।

React কম্পোনেন্ট হলো একটা সাধারণ জাভাস্ক্রিপ্ট ফাংশন, যা স্ক্রিনে ব্যবহারকারীকে দেখানোর জন্য কিছু মার্কআপ রিটার্ন করে:

```
functionMyButton() {
return <button>আমি একটা বাটন</button>;
}
```

এখন তুমি এই `MyButton` কম্পোনেন্টটাকে অন্য আরেকটা বড় কম্পোনেন্টের ভেতরে বসিয়ে দিতে পারো:

```
exportdefaultfunctionMyApp() {
return (
    <div>
      <h1>আমার অ্যাপে স্বাগতম!</h1>
      <MyButton />
    </div>
  );
}
```

গুরুত্বপূর্ণ কথা: খেয়াল করেছো, `<MyButton />`-এর নামটা **বড় হাতের M অক্ষর** দিয়ে শুরু হয়েছে? এটাই React কম্পোনেন্ট চেনার উপায়। **React কম্পোনেন্টের নাম সবসময় বড় হাতের অক্ষর দিয়ে শুরু করতে হয়**, আর সাধারণ HTML ট্যাগ (`<div>`, `<p>`) ছোট হাতের হয়।

`export default` কীওয়ার্ডটি দিয়ে বোঝানো হয় যে, এই ফাইলের প্রধান কম্পোনেন্ট কোনটি। জাভাস্ক্রিপ্টের এই সিনট্যাক্সগুলো নতুন মনে হলে [MDN (opens in a new tab)]() বা [javascript.info (opens in a new tab)]() থেকে দারুণভাবে শিখে নিতে পারো।

JSX দিয়ে মার্কআপ লেখা

উপরে যে HTML-এর মতো দেখতে কোড লিখলে, তাকে বলে **JSX**। এটা ব্যবহার করা বাধ্যতামূলক না, তবে বেশিরভাগ React ডেভেলপার এটা ব্যবহার করে কারণ এটা কোড লেখা অনেক সহজ করে দেয়।

JSX সাধারণ HTML-এর চেয়ে একটু বেশি কড়া। তোমাকে `<br>`-এর মতো ট্যাগগুলো `<br />` এভাবে বন্ধ করতে হয়। একটা কম্পোনেন্ট থেকে একাধিক JSX ট্যাগ সরাসরি রিটার্ন করা যায় না। সেগুলোকে অবশ্যই একটা প্যারেন্ট ট্যাগের (`<div>...</div>`) বা একটা অদৃশ্য র‍্যাপারের (`<>...</>`) ভেতরে মুড়িয়ে দিতে হয়।

```
functionAboutPage() {
return (
// এই <>...</> একটা অদৃশ্য বাক্স
    <>
      <h1>আমার সম্পর্কে</h1>
      <p>
        হ্যালো বন্ধুরা,
        <br />
        কেমন আছো?
      </p>
    </>
  );
}
```

পুরনো HTML কোডকে JSX-এ রূপান্তর করতে চাইলে অনলাইন কনভার্টার (opens in a new tab) ব্যবহার করতে পারো।

স্টাইল যোগ করা

React-এ স্টাইল যোগ করতে আমরা HTML-এর `class` অ্যাট্রিবিউটের বদলে `className` ব্যবহার করি। কারণ, `class` শব্দটি জাভাস্ক্রিপ্টের একটি নিজস্ব (রিজার্ভড) শব্দ।

```
<imgclassName="avatar" />
```

আর তারপর, তোমার CSS ফাইলে (`styles.css`) এই ক্লাসের জন্য স্টাইল লেখো:

```
/* তোমার CSS ফাইলে */
.avatar {
border-radius:50%;
}
```

তুমি চাইলে সরাসরি JSX-এর ভেতরে CSS লিখে ইনলাইন স্টাইলও দিতে পারো। তবে খেয়াল রেখো — এটা HTML-এর মতো না। এখানে CSS কোডটা **একটা জাভাস্ক্রিপ্ট অবজেক্ট** হিসেবে দিতে হয়।

```
conststyleObject= {
  border:"2px dashed red",
  backgroundColor:"lightyellow",// CSS-এর background-color → camelCase এ
backgroundColor
};

return <divstyle={styleObject}>আমি ইনলাইন স্টাইলড!</div>;
```

অথবা সরাসরি JSX-এর মধ্যে:

```
<divstyle={{ border:"1px solid gray", padding:"10px" }}>
  ইনলাইন স্টাইলিং শিখছি!
</div>
```

❗ নিয়মগুলো মনে রাখো:

1. CSS প্রপার্টির নামগুলো **camelCase** করতে হয়। যেমনঃ `background-color` → `backgroundColor`, `font-size` → `fontSize`
2. স্টাইলের ভ্যালুগুলো **স্ট্রিং** হিসেবে দিতে হয় (যেমন `"10px"`, `"red"`)
3. JSX-এ `style={{}}` মানে হলো — বাইরের `{}` দিয়ে JSX থেকে জাভাস্ক্রিপ্টে ঢুকছি, আর ভেতরের `{}` হচ্ছে জাভাস্ক্রিপ্ট অবজেক্ট।

কখন ইনলাইন স্টাইল ব্যবহার করবে?

- যদি স্টাইলটা খুব ছোট হয় (যেমন margin, padding, border ইত্যাদি)
- যদি ডায়নামিক মান বসাতে চাও (যেমন: condition অনুযায়ী background রঙ)

অসাধারণভাবে লিখেছো! নিচে আমি **আরও বিস্তারিত এবং সহজ ভাষায়** প্রতিটি অংশ ব্যাখ্যা করে দিলাম, যেন React-এ JSX ব্যবহার করে কীভাবে ডেটা দেখানো যায়, সেটা তুমি বা তোমার পাঠক একদম স্পষ্টভাবে বুঝতে পারো।

📦 ডেটা দেখানো (`displaying-data`)

JSX (JavaScript XML)-এর সবচেয়ে মজার বিষয় হলো: **তুমি এর ভেতরে সরাসরি জাভাস্ক্রিপ্ট কোড লিখতে পারো।** এই কাজটা করা হয় `{}` — কার্লি ব্র্যাকেটের মাধ্যমে।

এটা এমন, যেন JSX-এর ভেতরে একটা ছোট্ট দরজা খুলে, তুমি জাভাস্ক্রিপ্টের জগতে ঢুকে পড়ছো। এইভাবে তুমি ভ্যারিয়েবল, এক্সপ্রেশন, এমনকি ফাংশনের রিটার্ন ভ্যালুও দেখাতে পারো।

🎯 কার্লি ব্র্যাকেট `{}` দিয়ে কী হয়?

যেমন ধরো, তোমার একটা ইউজার অবজেক্ট আছে:

```
constuser= {
  name:"Hedy Lamarr",
};
```

তুমি চাইছো, সেই `name` টা `<h1>` ট্যাগে দেখাতে:

```
return <h1>{user.name} // user অবজেক্টের name প্রপার্টি দেখানো হচ্ছে</h1>;
```

👉 এখানে `{user.name}` মানে হলো — JSX-এ HTML-এর মতো দেখতে ট্যাগের ভিতরে **JavaScript ভ্যারিয়েবল** বসিয়ে দেওয়া।

JSX-এর অ্যাট্রিবিউটেও `{}` ব্যবহার করা যায়

JSX-এ HTML-এর মতোই অ্যাট্রিবিউট (যেমন `src`, `alt`, `style` ইত্যাদি) থাকে। কিন্তু যদি তুমি অ্যাট্রিবিউটে জাভাস্ক্রিপ্ট ভ্যালু বসাতে চাও, তাহলে সেটাও `{}` দিয়ে করতে হবে।

```
<imgsrc="user.imageUrl" /> // এটা src-এ ঠিকভাবে ভ্যালু দেবে না

<imgsrc={user.imageUrl} /> // এখানে React বুঝবে, এটা ভ্যারিয়েবল

consta=5;
constb=10;

return (
  <div>
    {/* সরাসরি গাণিতিক হিসাব */}
    <p>5 + 10 = {a + b}</p>

    {/* স্ট্রিং যোগ */}
    <p>{"Hello "+"World"}</p>

    {/* একটা জাভাস্ক্রিপ্ট এক্সপ্রেশন */}
    <p>{a * b}</p>
  </div>
);
```

এখানে `{a + b}`, `{"Hello " + "World"}`, `{a * b}` সবই এক্সপ্রেশন। এগুলো JSX-এ সরাসরি ভ্যালু হিসেবে রেন্ডার হবে।

উদাহরণ: JSX-এ ফাংশনের রিটার্ন ভ্যালু ব্যবহার

তুমি ফাংশন কলও করতে পারো JSX-এ, আর তার রিটার্ন ভ্যালু দেখাতে পারো।

```
functiongreet(name) {
return"Hello, "+ name +"!";
}

exportdefaultfunctionProfile() {
constuserName="Mojnu";

return (
    <div>
      <h1>{greet(userName)}</h1>
    </div>
  );
}
```

এখানে `greet(userName)` কল হচ্ছে JSX-এর মধ্যে, আর ফাংশনের রিটার্ন `"Hello, Mojnu!"` সরাসরি `<h1>` ট্যাগের মধ্যে দেখাবে।

```
constuser= {
  name:"Mojnu",
  age:26,
};

functiongetYearOfBirth(age) {
constcurrentYear=newDate().getFullYear();
return currentYear - age;
}

exportdefaultfunctionProfile() {
return (
    <div>
      <h1>{user.name}</h1>
      <p>Born in {getYearOfBirth(user.age)}</p>
      <p>Next year, you will be {user.age +1} years old.</p>
    </div>
  );
}
```

- `{getYearOfBirth(user.age)}` — ফাংশন কল করে তার রিটার্ন দেখাচ্ছে
- `{user.age + 1}` — এক্সপ্রেশন হিসেবে বয়সের ওপর গাণিতিক অপারেশন

শর্ত দিয়ে কিছু দেখানো

React-এ শর্তসাপেক্ষে কিছু দেখানোর জন্য আলাদা কোনো সিনট্যাক্স নেই। তুমি সাধারণ জাভাস্ক্রিপ্টের মতোই `if` স্টেটমেন্ট, কন্ডিশনাল `?` অপারেটর, অথবা লজিক্যাল `&&` অপারেটর ব্যবহার করতে পারো।

```
// if স্টেটমেন্ট ব্যবহার করে
let content;
if (isLoggedIn) {
  content = <AdminPanel />;
} else {
  content = <LoginForm />;
}
return <div>{content}</div>;

// অথবা আরও স্মার্ট ভাবে, কন্ডিশনাল (?) অপারেটর দিয়ে
return <div>{isLoggedIn ? <AdminPanel /> : <LoginForm />}</div>;

// যখন শুধু if লাগে (else দরকার নেই), && অপারেটর দিয়ে
return <div>{isLoggedIn && <AdminPanel />}</div>;
```

তালিকা দেখানো

কোনো তালিকা বা অ্যারে দেখানোর জন্য তুমি জাভাস্ক্রিপ্টের `map()` ফাংশন ব্যবহার করতে পারো।

```
constproducts= [
  { title:"ফুলকপি", id:1 },
  { title:"রসুন", id:2 },
  { title:"পেঁপে", id:3 },
];

constlistItems=products.map((product) => (
  <likey={product.id}>{product.title}</li>
));

return <ul>{listItems}</ul>;
```

খুব জরুরি কথা: তালিকা দেখানোর সময় প্রত্যেকটা আইটেমের জন্য একটা ইউনিক `key` দিতেই হবে। এই `key` অনেকটা ক্লাসের রোল নম্বরের মতো। React এটা দিয়ে বুঝতে পারে তালিকার কোন আইটেম কোনটা, যা পারফরম্যান্সের জন্য খুব দরকারি। সাধারণত ডেটাবেসের ID-কে `key` হিসেবে ব্যবহার করা হয়।

ইউজারের কাজে সাড়া দেওয়া

ইউজারের বিভিন্ন কাজের (যেমন: ক্লিক) জন্য তুমি **ইভেন্ট হ্যান্ডলার** ফাংশন লিখতে পারো।

```
functionMyButton() {
// এটা হলো ইভেন্ট হ্যান্ডলার ফাংশন
functionhandleClick() {
alert("তুমি আমাকে ক্লিক করেছো!");
  }

return (
// onClick ইভেন্টে ফাংশনটা জুড়ে দিলাম
    <buttononClick={handleClick}>আমাকে ক্লিক করো</button>
  );
}
```

খেয়াল রাখবে, `onClick={handleClick}` লিখতে হবে, `onClick={handleClick()}` নয়। কারণ আমরা ফাংশনটাকে React-এর হাতে তুলে দিচ্ছি, যেন সে ক্লিকের পর কল করতে পারে; আমরা নিজে এখনই কল করছি না।

স্ক্রিন আপডেট করা

যদি কোনো কম্পোনেন্টকে কোনো তথ্য "মনে রাখতে" হয় (যেমন, একটা বাটন কতবার ক্লিক করা হলো), তখন আমরা ব্যবহার করি **স্টেট (State)**। স্টেট হলো একটা কম্পোনেন্টের মেমরি।

স্টেট ব্যবহার করার জন্য React আমাদের `useState` নামে একটা বিশেষ ফাংশন বা **হুক (Hook)** দেয়।

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

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

functionhandleClick() {
setCount(count +1);
  }

return <buttononClick={handleClick}>ক্লিক হয়েছে {count} বার</button>;
}
```

`useState(0)` আমাদের দুটো জিনিস দেয়:

1. **`count`**: বর্তমান স্টেট (যার প্রাথমিক মান `0`)।
2. **`setCount`**: স্টেট পরিবর্তন করার জন্য একটি ফাংশন।

যখনই `setCount` কল হয়, React কম্পোনেন্টটাকে নতুন ডেটা দিয়ে আবার রেন্ডার করে এবং স্ক্রিনে আপডেট দেখায়। তুমি যদি একই কম্পোনেন্ট একাধিকবার ব্যবহার করো, প্রত্যেকটা তার নিজস্ব স্টেট আলাদাভাবে মনে রাখবে।

হুকস ব্যবহার করা

`use` দিয়ে শুরু হওয়া ফাংশনগুলোকে **হুকস (Hooks)** বলে। `useState` হলো React-এর একটি বিল্ট-ইন হুক। হুকস ব্যবহারের কিছু নিয়ম আছে, যেমন—এগুলোকে শুধুমাত্র কম্পোনেন্টের একদম শুরুতে (টপ-লেভেলে) কল করা যায়, কোনো লুপ বা কন্ডিশনের ভেতরে নয়।

কম্পোনেন্টগুলোর মধ্যে ডেটা শেয়ার করা

অনেক সময় একাধিক কম্পোনেন্টের একই ডেটা ব্যবহার করার এবং একসাথে আপডেট হওয়ার দরকার হয়। এর জন্য, আমরা স্টেটকে চাইল্ড কম্পোনেন্ট থেকে বের করে তাদের সাধারণ প্যারেন্ট কম্পোনেন্টে নিয়ে যাই। এই পদ্ধতিকে বলে **লিফটিং স্টেট আপ (Lifting State Up)**।

এরপর প্যারেন্ট কম্পোনেন্ট সেই স্টেট এবং স্টেট পরিবর্তনের ফাংশন তার চাইল্ড কম্পোনেন্টগুলোতে **প্রপস (props)** হিসেবে পাস করে দেয়।

ধাপ ১: স্টেটকে প্যারেন্টে (`MyApp`) নিয়ে আসা।

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

functionhandleClick() {
setCount(count +1);
  }
// ...
}
```

ধাপ ২: প্যারেন্ট থেকে চাইল্ডে স্টেট ও ফাংশনকে `props` হিসেবে পাস করা।

```
exportdefaultfunctionMyApp() {
// ... স্টেট ও ফাংশন এখানে আছে
return (
    <div>
      <h1>একসাথে আপডেট হওয়া কাউন্টার</h1>
      <MyButtoncount={count} onClick={handleClick} />
      <MyButtoncount={count} onClick={handleClick} />
    </div>
  );
}
```

ধাপ ৩: চাইল্ড কম্পোনেন্টে (`MyButton`) সেই `props` গ্রহণ ও ব্যবহার করা।

```
functionMyButton({ count, onClick }) {
return <buttononClick={onClick}>ক্লিক হয়েছে {count} বার</button>;
}
```

এখন যেকোনো একটা বাটনে ক্লিক করলেই প্যারেন্টের স্টেট আপডেট হবে এবং সেই আপডেট হওয়া মান দুটো চাইল্ডই স্ক্রিনে দেখাবে।

এরপর কী?

ব্যস! তুমি কিন্তু React-এর সবচেয়ে দরকারি বিষয়গুলো শিখে ফেলেছো। এগুলোই মূল ভিত্তি।

এবার শুধু পড়লে হবে না, নিজে হাতে কোড করতে হবে। আমার পরামর্শ হলো, তুমি **React-এর অফিশিয়াল টিক-ট্যাক-টো টিউটোরিয়ালটা (opens in a new tab)** করে ফেলো। দেখবে, এই সবকিছু ব্যবহার করে একটা গেম বানালে তোমার আত্মবিশ্বাস অনেক বেড়ে যাবে!

💙 স্পনসর করুন

আপনি আমার কাজকে মূল্যায়ন করেন 😊

- মাসিক নিউজলেটার আপডেট পান
- স্পনসর ব্যাজ এই সাইটে প্রদর্শিত হবে
- নতুন ফিচার আনার জন্য প্রস্তাব দেওয়ার সুযোগ
- এই সাইটে বিজ্ঞাপন দিতে পারবেন