**📘 React Hooks Summary**

**1️⃣ useState() Hook**

**✅ التعريف**

* Hook لإنشاء **state variable** يحتفظ بقيمته بين كل عملية render وأخرى.
* أي تغيير على الـ state → يعمل re-render → UI يتحدث مباشرة.

**⚙️ آلية العمل**

1. useState(initialValue) بيرجع:
   * قيمة state
   * setter function لتحديثها
2. عند استدعاء setter:
   * React تقارن القيمة الجديدة بالقديمة
   * إذا اختلفت → تعمل re-render

**🎯 الهدف**

* ربط البيانات مع الـ UI بشكل ديناميكي.
* أي تعديل في البيانات يظهر فورًا.

**✨ مثال:**

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

<button onClick={() => setCount(count + 1)}>+</button>

**🔑 ملاحظات مهمة**

1. **القيمة الابتدائية** تستخدم فقط عند أول render.
2. ممكن تكون القيمة: Number, String, Boolean, Array, Object, Function.
3. التحديث **Asynchronous** (غير فوري).
4. لو التحديث يعتمد على القيمة السابقة → استعملي callback:
5. setCount(prev => prev + 1);
6. إذا القيمة نفسها → ما بصير re-render.
7. لازم تستدعي useState **داخل Component function أو Custom Hook فقط**.
8. React بتعمل **Batching updates** لتجميع التحديثات.
9. مع Objects/Arrays → لازم تستخدمي نسخة جديدة (immutability).

**2️⃣ useEffect() Hook**

**✅ التعريف**

* Hook لإدارة **Side Effects** (أشياء تحصل خارج الـ render).
* أمثلة: جلب بيانات من API، التفاعل مع DOM، timers، subscriptions.

**⚙️ الصيغة:**

useEffect(() => {

// setup logic

return () => {

// cleanup logic

};

}, [dependencies]);

**🎯 الاستخدامات**

* [] → مرة وحدة عند mount.
* [x, y] → كل ما تتغير x أو y.
* بدون [] → بعد كل render.

**✨ أمثلة:**

// ⏳ Mount مرة وحدة

useEffect(() => console.log("Mounted"), []);

// 📡 عند تغير count

useEffect(() => console.log(count), [count]);

// 🧹 Cleanup مع interval

useEffect(() => {

const timer = setInterval(() => console.log("tick"), 1000);

return () => clearInterval(timer);

}, []);

**🔑 النقاط المهمة**

* يتنفذ دائمًا بعد render (مش أثناءه).
* لازم تضيف كل المتغيرات المستخدمة بالـ effect داخل dependencies.
* ممكن يكون عندك أكثر من useEffect بنفس الـ component.
* **Cleanup** مهم عشان تمنع memory leaks.

**3️⃣ الفرق بين useEffect و useLayoutEffect**

* **useEffect**:
  + React يسمح للمتصفح يرسم (paint) الأول، بعدين ينفذ الكود.
  + مناسب للأشياء الغير مرئية للمستخدم مباشرة (API calls, logs).
* **useLayoutEffect**:
  + React يوقف الرسم → ينفذ الكود → بعدين يرسم.
  + مناسب لأشياء مرتبطة مباشرة بالـ UI (قياسات DOM, scroll, tooltip).
* ❌ لو استخدمتي useEffect غلط → ممكن يصير flicker (وميض).

**4️⃣ CSR vs SSR vs Hydration**

* **CSR (Client-Side Rendering)**:
  + السيرفر يرجع HTML شبه فاضي + ملفات JS.
  + React بالمتصفح تبني UI كامل.
  + أبطأ بالتحميل الأول، مش جيد للـ SEO.
* **SSR (Server-Side Rendering)**:
  + السيرفر يرجع HTML جاهز بالكومبوننتات.
  + المستخدم يشوف صفحة بسرعة.
  + بعدين React بالمتصفح يعمل **Hydration** (ربط الأحداث والـ state).
* **Hydration**:
  + المتصفح ما بيعيد بناء HTML.
  + React بس "يحيي" (activate) العناصر الموجودة.

**5️⃣ مشاكل محتملة مع useEffect**

**🌀 Infinite Loop**

* يحصل إذا:
  1. الـ Effect يحدث state.
  2. التحديث يؤدي لتغير dependencies.
* **الحل**:
  1. فكري إذا فعلاً لازم تحدثي state.
  2. لو بيانات مش مرتبطة بالـ render → استعملي useRef بدل state.

**🧹 Cleanup Runs Unexpectedly**

* الـ cleanup يشتغل:
  + قبل كل re-render لما تتغير dependencies.
  + وعند unmount.
  + وفي وضع التطوير (Strict Mode) → React يشغل setup + cleanup مرتين للتأكد من أمان الكود.
* **قاعدة**: cleanup لازم يكون لإلغاء شيء أنشأه setup (event listener, timer…).

**📌 الخلاصة**

* useState لإدارة البيانات الداخلية للـ component.
* useEffect لإدارة الـ Side Effects (API, timers, listeners…).
* useLayoutEffect لما لازم تمنعي flicker وتعدلي DOM قبل الرسم.
* CSR = Rendering بالمتصفح، SSR = Rendering بالسيرفر + Hydration بالمتصفح.
* React Strict Mode يخلي cleanup يشتغل زيادة (فقط بالـ development).