

# 4차시 스크립트

## Recap

### 1. React의 핵심 개념

#### State와 리렌더링

- **State**는 컴포넌트의 **UI 상태를 저장하는 변수**이다.
- 일반 변수와 달리 **리렌더링에도 값이 유지**되며, 값이 변경되면 해당 컴포넌트가 자동으로 **다시 그려짐 (re-rendering)**.
- 예: 버튼 클릭 수, 입력값, API 응답 데이터 등.
- **단점**: 새로고침하면 state는 초기화된다 → **브라우저 메모리에 저장**되기 때문.

#### State vs 일반 변수

구분	일반 변수	State
리렌더링 후 값 유지	✗	✓
UI 자동 갱신	✗	✓
선언 위치	함수 내부 어디든 가능	<code>useState</code> 사용
예시	<code>let count = 0;</code>	<code>const [count, setCount] = useState(0);</code>

#### 왜 state가 중요한가?

- 예: 버튼 클릭 시 드롭다운이 **의도치 않게 닫히는 문제** 발생 가능 → 드롭다운의 열림 여부도 state로 관리하면 해결 가능

### 2. React의 렌더링 방식

#### Virtual DOM

- 실제 DOM을 직접 조작하지 않고, **메모리에 복사된 Virtual DOM**을 사용
- 변경된 부분만 실제 DOM에 적용하여 성능 향상 (diffing 알고리즘)

#### 선언형 UI

- `document.querySelector()` 와 같은 명령형 방식 대신, **UI를 데이터 기반으로 선언**한다

- 코드가 더 예측 가능하고 유지보수 쉬움

### 3. 컴포넌트란?

- UI를 구성하는 재사용 가능한 코드 조각
- 목적: 유지보수성, 재사용성, 코드 분리
- 컴포넌트는 **\*\*함수(function)\*\***로 정의 (또는 class-based 컴포넌트 – 이제는 잘 안 씀)

예:

```
function Button({ label }) {  
  return <button>{label}</button>;  
}
```

### 4. React 프로젝트 생성 (with Vite)

```
npm create vite@latest my-app --template react  
cd my-app  
npm install  
npm run dev
```

| Vite는 빠른 개발 서버와 빠른 빌드를 제공하는 도구

### 5. 프로젝트 구조

```
my-app/  
├── node_modules/  # 설치된 라이브러리  
├── public/        # 정적 자원 (index.html 등)  
├── src/           # 실제 코드 작성 공간  
│   └── App.js     # 최상위 컴포넌트
```

```

|   |— index.js    # 앱 진입점 (root 렌더링)
|   |— components/ # 사용자 정의 컴포넌트
|   |— assets/     # 이미지, 폰트 등 자원
|— package.json    # 의존성 및 스크립트 정의
|— README.md       # 프로젝트 설명

```

## useState, useEffect와 API 실습

### 1. React Hooks란?

- 함수형 컴포넌트에서 상태(state)와 생명주기(lifecycle)를 다룰 수 있게 해주는 함수들
- 기존에는 상태관리나 생명주기를 클래스 컴포넌트에서만 할 수 있었음
- 대표적인 Hook: ( 지금은 useState, useEffect만 알아도 됨)
  - `useState` → 상태를 만든다
  - `useEffect` → 생명주기 & 비동기 처리
  - `useRef` → DOM 참조
  - `useContext` → 전역 상태
  - `useReducer` → 복잡한 상태 로직
  - `useMemo`, `useCallback` → 최적화 관련

Hook은 컴포넌트의 최상단에서만 호출되어야 하며, 조건문/반복문 안에서 호출하면 안 됨 (React의 규칙)

### 2. useEffect에서 자주 쓰는 패턴

목적	코드 예시
마운트 시 1회 실행	<code>useEffect(() =&gt; { ... }, [])</code>
특정 값이 변경될 때 실행	<code>useEffect(() =&gt; { ... }, [value])</code>
언마운트(clean-up)	<code>useEffect(() =&gt; { return () =&gt; {...}; }, [])</code>

## 상태 관리 고도화 개념

### 3. 비동기 처리와 상태 세분화

- API 호출 상태를 나누면 UX가 좋아짐:
  - `loading` (불러오는 중)
  - `error` (에러 발생)
  - `data` (성공적 응답)

```
const [loading, setLoading] = useState(true);
const [error, setError] = useState(null);
const [data, setData] = useState([]);

useEffect(() => {
  fetch(...)
    .then(res => res.json())
    .then(data => setData(data))
    .catch(err => setError(err))
    .finally(() => setLoading(false));
}, []);
```

#### 1 `useEffect` 와 생명주기 (Lifecycle)

##### 생명주기란?

컴포넌트가 화면에 **등장(mount)** → **변경(update)** → **사라질 때(unmount)** 까지의 흐름

##### `useEffect` 란?

- 함수형 컴포넌트에서 생명주기 역할을 하는 **React Hook**
- 특정 조건일 때만 실행 가능
- API 호출, 구독 설정, 타이머 설정 등 "사이드 이펙트(side effects)"를 처리할 때 사용

```
useEffect(() => {
  // 이 안의 코드는 컴포넌트가 처음 렌더링될 때 실행됨 (mount)
}, []);
```

생명주기 단계

클래스 컴포넌트 메서드

함수 컴포넌트 with `useEffect`

마운트 시	componentDidMount	<code>useEffect(() =&gt; {}, [])</code>
업데이트 시	componentDidUpdate	<code>useEffect(() =&gt; {}, [deps])</code>
언마운트 시	componentWillUnmount	<code>useEffect(() =&gt; return () =&gt; {...}, [])</code>

## 2 부모-자식 간 데이터 전달 (Props & Lifting State Up)

### Props

- 부모 → 자식으로 데이터 전달
- 읽기 전용 (자식은 props를 직접 바꿀 수 없음)

```
function Child({ name }) {
  return <p>안녕, {name}!</p>;
}

<Child name="철수" />
```

### ↑ Lifting State Up (상태 끌어올리기)

- 형제 컴포넌트끼리 데이터를 공유하고 싶다면, 공통 부모 컴포넌트에 state를 올려야 함

```
// 부모 컴포넌트
function Parent() {
  const [count, setCount] = useState(0);
  return (
    <>
      <ChildA count={count} />
      <ChildB onClick={() => setCount(count + 1)} />
    </>
  );
}
```

## 3 API 요청 (fetch / axios)

### 데이터 가져오기

```
// fetch 예제
useEffect(() => {
  fetch("https://www.themealdb.com/api/json/v1/1/search.php?s=chicken")
    .then(res => res.json())
    .then(data => console.log(data));
}, []);
```

```
// axios 예제
import axios from 'axios';

useEffect(() => {
  axios.get("https://www.themealdb.com/api/json/v1/1/search.php?s=chicken")
    .then(res => console.log(res.data));
}, []);
```

#### 4 실습 예제: TheMealDB API 렌더링

##### 목표

- 사용자 입력을 받아 API에서 데이터를 받아오기
- 받아온 데이터를 리스트 형태로 출력하기

```
function App() {
  const [meals, setMeals] = useState([]);

  useEffect(() => {
    fetch("https://www.themealdb.com/api/json/v1/1/search.php?s=chicken")
      .then(res => res.json())
      .then(data => setMeals(data.meals));
  }, []);

  return (
    <div>
      <h1>🍗 Meal List</h1>
```

```
<ul>
  {meals.map(meal => (
    <li key={meal.idMeal}>{meal.strMeal}</li>
  ))}
</ul>
</div>
);
}
```