

# 처음 배우는 훈련생을 위한 react 가이드

## 1. 1~2주차: JavaScript 보강 (60시간)

### 가. Week 1 - JS 기본기 다지기

- ▶ 자바스크립트 소개 및 개발환경 점검
- ▶ 변수와 자료형 (let, const, var, Number, String, Boolean 등)
- ▶ 함수와 스코프 (선언, 표현식, 화살표 함수)
- ▶ 객체와 배열 기초
- ▶ 배열 메서드 (map, filter, reduce)
- ▶ ES6 문법 (템플릿 리터럴, 구조분해, 스프레드 등)
- ▶ this와 바인딩 이해
- ▶ 종합 실습: ToDo 리스트 CRUD 구현
- ▶ 주간 리뷰 & 퀴즈

### 나. Week 2 - DOM & 브라우저 API

- ▶ DOM(Document Object Model) 이해
- ▶ DOM 요소 선택과 조작 (innerText, innerHTML, classList, style)
- ▶ 이벤트 핸들링 (addEventListener, 이벤트 객체)
- ▶ localStorage 활용 (데이터 저장/불러오기, JSON.stringify/parse)
- ▶ fetch API 기초 (HTTP 요청/응답, 공공 API 연동)
- ▶ JSON 데이터 다루기 (형식, 변환)
- ▶ async/await 비동기 처리 및 에러 핸들링
- ▶ 미니 프로젝트: 간단한 영화 검색 앱

## 2. 3~4주차: React 기초 & 심화 (60시간)

### 가. Week 3 - React 기초

- ▶ Vite + React 환경 세팅
- ▶ JSX 문법과 표현식
- ▶ 컴포넌트 구조와 Props
- ▶ State 개념과 활용
- ▶ 이벤트 핸들링 in React
- ▶ 조건부 렌더링 & 리스트 렌더링(map)
- ▶ 실습: 카운터 앱, 출석부 앱

### 나. Week 4 - React 심화 + API 연동

- ▶ useEffect 개념과 데이터 가져오기
- ▶ fetch API 활용 in React
- ▶ 로딩/에러 처리 패턴
- ▶ 외부 API 실습 (날씨, 영화)
- ▶ JSON 데이터 테이블 출력

- ▶ JSON CRUD 시뮬레이션
- ▶ 종합 실습: API 기반 리스트 앱

### 3. 5주차: 데이터 시각화 (40시간)

#### 가. Week 5 - Chart.js & Recharts

- ▶ Chart.js 소개 및 설치
- ▶ 기본 차트: 막대, 선, 파이 그래프
- ▶ Recharts 소개 및 실습 (BarChart, LineChart, PieChart)
- ▶ 차트 커스터마이징 (색상, 라벨, 툴팁 등)
- ▶ 종합 실습: 지출/수입 대시보드 프로젝트

### 4. 6~7주차: Firebase (80시간)

#### 가. Week 6 - Firebase 기초 (Firestore CRUD)

- ▶ Firebase 소개 및 프로젝트 생성
- ▶ SDK 연결과 보안 규칙 이해
- ▶ Firestore 구조 설계
- ▶ 데이터 읽기/쓰기/수정/삭제
- ▶ React + Firestore CRUD 연동
- ▶ 미니 프로젝트: 출석 관리 DB 앱

#### 나. Week 7 - Firebase 심화 (Auth, Storage, Hosting)

- ▶ Firebase Authentication (회원가입, 로그인)
- ▶ React와 Auth 상태 관리
- ▶ Firestore + Auth 연계
- ▶ Firebase Storage (이미지 업로드)
- ▶ Hosting (Firebase 배포)
- ▶ 종합 실습: 출석·성적 관리 웹앱

| React 7주 완성 세부 시간표 |                        |                  |                       |                          |                       |
|--------------------|------------------------|------------------|-----------------------|--------------------------|-----------------------|
|                    | 9월 30일                 | 10월 1일           | 10월 2일                | 10월 10일                  | 10월 13일               |
| 오전                 | JS 기본 (let/const, 자료형) | 함수/스코프           | 객체/배열 기초              | 배열 메서드 map/filter, 고차 함수 | JS 미니 프로젝트(ToDo CRUD) |
| 오후                 | ES6 문법(화살표 함수, 템플릿)    | this & 바인딩       | 구조분해/스프레드             |                          |                       |
| 10월 14일            |                        | 10월 15일          | 10월 16일               | 10월 17일                  | 10월 20일               |
| 오전                 | DOM 조작(querySelector)  | localStorage 활용  | JSON(parse/stringify) | Vite + React 환경 세팅       | JSX 문법                |
| 오후                 | 이벤트(addEventListener)  | fetch API 실습(날씨) | async/await           | Hello React 실습           | JSX 표현식/조건부 렌더링       |
| 10월 21일            |                        | 10월 22일          | 10월 23일               | 10월 24일                  | 10월 27일               |
| 오전                 | 컴포넌트 & Props           | State 개념         | 이벤트 핸들링               | useEffect 소개             | fetch API (React)     |
| 오후                 | 리스트 렌더링(map)           | 카운터 앱            | 미니 프로젝트(출석부)          | JSON CRUD 시뮬             | API 실습(영화)            |
| 10월 28일            |                        | 10월 29일          | 10월 30일               | 10월 31일                  | 11월 3일                |
| 오전                 | 로딩/에러 처리               | 외부 API 실습(날씨)    | JSON 데이터 출력           | Chart.js 소개 & 설치         | 막대 그래프                |
| 오후                 | 조건부 렌더링 실습             | JSON + 테이블 렌더링   | 미니 프로젝트(API 기반 리스트)   | Recharts 소개              | BarChart 실습           |
| 11월 4일             |                        | 11월 5일           | 11월 6일                | 11월 7일                   | 11월 10일               |
| 오전                 | 선 그래프                  | 파이 차트            | 종합 차트 실습              | Firebase 소개/세팅           | Firestore 구조          |
| 오후                 | LineChart 실습           | PieChart 실습      | 미니 프로젝트(지출 대시보드)      | SDK 연결                   | React + Firestore 연동  |
| 11월 11일            |                        | 11월 12일          | 11월 13일               | 11월 14일                  | 11월 17일               |
| 오전                 | 데이터 읽기                 | 데이터 쓰기           | Firebase 콘솔 실습        | Auth 개요                  | 회원가입/로그인 구현           |
| 오후                 | CRUD 추가                | CRUD 수정/삭제       | 미니 프로젝트(출석 DB)        | React + Auth 상태 관리       | Auth + Firestore 연동   |
| 11월 18일            |                        | 11월 19일          | 11월 20일               |                          |                       |
| 오전                 | Storage 소개             | 이미지 업로드          | Hosting 배포            |                          |                       |
| 오후                 | Storage + DB 연계        | CRUD 종합 실습       | 미니 프로젝트(출석·성적 관리 앱)   |                          |                       |

## - Day 1 -

### 1. 자바스크립트 소개

- 웹 페이지를 동적으로 만들기 위해 사용하는 대표 프로그래밍 언어
- HTML → 구조, CSS → 디자인/스타일, JS → 동작/기능
- 현재는 브라우저뿐만 아니라 Node.js 환경을 통해 서버에서도 사용됨
- 특징
  - HTML 문서내에 포함되어 해석됨
  - 이벤트에 반응하여 처리가 가능함
  - 브라우저 프로그래밍을 통해 DB정보와 전자상거래, CGI를 대체할 수 있음
  - 클래스 기반의 프로그래밍 가능
  - 자바스크립트의 라인구분 : 세미콜론(;) 사용
  - 변수 선언을 하지 않아도 바로 사용 가능
  - 사용자 이벤트 제어 가능
  - DOM 사용 가능
  - 클라이언트 측 객체지향 스크립트 언어
- 자바스크립트의 활용
  - 자바스크립트의 삽입 : 인라인 형태, 스크립트 태그를 사용하는 형태, 외부 파일을 지정하는 방법
    - 인라인 형태 : 이벤트와 함께 사용
      - <div onMouseOver="document.bgColor='orange'">
      - <a href="javascript:document.write('test')"> test </a>
    - 스크립트 태그를 사용하는 방법
      - 1 -- <script language="javascript"> </script>
      - <script type="text/javascript"> </script>
    - 외부 파일을 지정하는 방법
      - <script language="javascript" src="스크립트파일.js"> </script>
    - 주석처리 : (한줄주석) //, (여러라인 주석) /\* ... \*/, (HTML 주석) <!-- ... -->

### 2. JavaScript의 데이터 타입

| 구분 | 데이터 타입    | 내용              |
|----|-----------|-----------------|
| 1  | number    | 정수, 실수등의 숫자 타입  |
| 2  | boolean   | true, false를 표시 |
| 3  | string    | 문자열 표시          |
| 4  | object    | 복합데이터 타입        |
| 5  | undefined | 정의되지 않은 타입      |

### 3. 변수 선언

- 변수 : 메모리에 데이터를 저장할 수 있도록 저장공간을 확보하는 것
- 변수가 메모리에 할당되면, 그 공간은 같은 타입의 데이터를 지속적으로 변경하여 저장할 수 있음.
- 변수는 임의의 장소에 할당되며, 숫자형 주소가 지정됨.
- JavaScript는 변수의 타입검사 등을 하지 않는다.
- 변수의 선언이 없이 사용을 해도 된다.
- 자바스크립트에서 변수를 선언하는 방법은 3가지가 있습니다.

```
let name = "홍길동"; // 변경 가능 (블록 스코프)
const age = 25; // 변경 불가 (상수)
var old = "옛날 방식"; // 함수 스코프 (권장 X)
```

let : 값 변경 가능, 블록 스코프 (가장 많이 사용)

const : 값 변경 불가, 반드시 선언과 동시에 초기화해야 함

var : 과거 방식, 함수 스코프라 버그 유발 가능 → 사용 지양

```
let a = 10;
a = 20; // 가능
const b = 30;
// b = 40; // 에러 발생 (const는 변경 불가)
```

### 4. 상수

- 상수 개념 : 메모리에 데이터를 저장할 수 있도록 저장공간을 확보하는 것
- 상수가 메모리에 생성되면 그 공간은 변경하여 저장할 수가 없음
- 상수의 종류 : 숫자형 상수, 문자형 상수, NaN(Not a Number), null
- 리터럴 : 표현 가능한 데이터( 10,203, 1.23, true, 'test string', '훈민정음' )

### 5. 자료형

- 원시 자료형 (Primitive Type)
- Number → 모든 숫자 (정수, 실수)
- String → 문자열
- Boolean → true/false
- null → “의도적으로 비어있음”
- undefined → 값이 할당되지 않음
- Symbol → 고유한 값 (잘 사용 안 함, 고급 주제)
- BigInt → 매우 큰 정수 표현

```
let num = 10; // Number
let text = "안녕하세요"; // String
let flag = true; // Boolean
```

```

let empty = null;          // null
let unknown;              // undefined (값을 안 넣었음)

console.log(typeof num);   // number
console.log(typeof text);  // string
console.log(typeof flag);  // boolean
console.log(typeof empty); // object (JS의 오래된 버그)
console.log(typeof unknown); // undefined

```

## 6. JavaScript 데이터 타입 변환

- Number(String str) : 문자열을 숫자로 변환
  - 지정된 매개변수가 숫자가 아닌 값이라면 NaN(Not a Number)값 출력
- String(숫자) : 숫자를 문자열로 변환
- 변수.toString() : 변수를 문자열로 변환
- parseInt(var str), parseFloat(var str) : 문자열을 숫자로 변환
  - 단, 숫자형 문자를 변환시키는 것이 의미가 있다. ( 123a → 123 )
- Boolean( 변환값 ) : 변환할 값이 0, NaN, "", null, undefined일 경우를 제외하고는 true

## 7. 연산자

- 산술 연산자

```

console.log(5 + 3); // 8
console.log(5 - 3); // 2
console.log(5 * 3); // 15
console.log(10 / 2); // 5
console.log(10 % 3); // 1 (나머지)

```

- 비교 연산자

```

console.log(5 == "5"); // true (값만 비교, 타입 무시)
console.log(5 === "5"); // false (값 + 타입 모두 비교)
console.log(10 != "10"); // false
console.log(10 !== "10");// true

```

### ■ 논리 연산자

```
console.log(true && false); // false (AND)
console.log(true || false); // true (OR)
console.log(!true); // false (NOT)
```

### ■ 조건 연산자(삼항연산자)

```
let result = 10 > 5 ? "5보다 크다" : "5보다 작다";
(조건) ? (A) : (B);
```

## 8. 템플릿문자열

### ■ 백틱(`)을 사용하면 문자열 안에 변수를 쉽게 넣을 수 있다

```
let name = "철수";
let age = 20;

console.log("안녕하세요. 저는 " + name + "이고 나이는 " + age + "살입니다.");
console.log(`안녕하세요. 저는 ${name}이고 나이는 ${age}살입니다.`); // 추천 방식
```

### ◆ 실습 문제

#### 1. BMI 계산기 만들기

- 변수 height(m), weight(kg)를 선언
- $BMI = \frac{weight}{height^2}$  계산
- BMI 값 출력

#### 2. 장바구니 합계 계산

```
let item1 = 12000;
let item2 = 3500;
let item3 = 8000;
```

- 총합을 구해 출력
- 만약 총합  $\geq 20000$  이면 "무료배송" 출력
- 아니면 "배송비 3000원 추가" 출력

#### 3. 짹수/홀수 판별기

- 변수 num에 숫자 하나 저장
- % 연산자를 이용해 "짝수" 또는 "홀수" 출력

#### 4. 숫자 문자열 더하기

- 변수 num1 = "10", num2 = "20" 선언
- 이 두 값을 더해서 "1020"이 출력되도록 하고
- 다시 숫자로 변환하여 합이 30이 출력되도록 하세요.

#### 5. 문자열 다루기 심화

- 변수 fullName = "홍길동"
- 템플릿 문자열을 이용해 "안녕하세요, 저는 홍길동입니다." 출력
- 문자열 길이를 출력 → "이름의 길이는 3입니다."
- 첫 글자만 추출해서 "성은 홍입니다." 출력
- 문자열을 거꾸로 뒤집어 "동길홍" 출력

## - Day 2 -

### 9. 함수 선언

- 특정 동작을 묶어서 재사용할 수 있게 만든 코드 블록
- return 키워드를 만나면 함수 실행이 종료되고 값을 반환

```
function add(a, b) {  
    return a + b;  
}  
console.log(add(2, 3)); // 5  
console.log(add(10, 20)); // 30
```

- 함수 선언은 호이스팅 되어 코드 위쪽에서 호출해도 실행 가능

```
console.log(square(4)); // 16 (함수 정의보다 위에서 호출 가능)
```

```
function square(x) {  
    return x * x;  
}
```

### 10. 스코프

- 변수의 유효 범위
- 어디서 변수를 사용할 수 있는지를 결정함
- 블록 스코프

```
let x = 10;  
  
if (true) {  
    let x = 20;  
    console.log(x); // 20  
}  
  
console.log(x); // 10
```

- 함수 스코프

```
function test() {  
    let y = 30;  
    console.log(y);  
}  
test();  
// console.log(y); // 에러 (함수 밖에서는 접근 불가)
```

## 11. 함수 표현식

- 함수도 값이 될 수 있음 → 변수에 저장 가능

```
const multiply = function(a, b) {  
    return a * b;  
};  
console.log(multiply(3, 4)); // 12
```

- 함수 표현식은 호이스팅이 되지 않음 → 반드시 정의 후 호출해야 함 (권장)

## 12. 화살표 함수

- ES6부터 추가된 함수 표현식 문법으로, 더 간결하게 작성할 수 있습니다

```
// 일반 함수  
function add(a, b) { return a + b; }  
// 화살표 함수  
const add = (a, b) => { return a + b; };
```

- 한 줄일 때는 return 생략 가능

```
const square = x => x * x;  
console.log(square(5)); // 25
```

- 매개변수가 없을 때

```
const hello = () => console.log("안녕하세요!");  
hello();
```

- 객체 반환 시 괄호 필요

```
// 객체를 바로 반환할 때는 ()로 감싸야 함
const makeUser = (name, age) => ({ name, age });
console.log(makeUser("철수", 20)); // { name: "철수", age: 20 }
```

- this와의 차이점
  - 일반 함수 : 호출 방식에 따라 this가 달라짐
  - 화살표 함수 : 자신을 감싸는 외부 스코프의 this를 그대로 사용

```
let user = {
  name : "홍길동",
  normalHello : function() {
    console.log("안녕 나는 " + this.name);
  },
  arrowHello : () => {
    console.log("안녕 나는 " + this.name);
  }
};
user.normalHello(); // "안녕 나는 홍길동"
user.arrowHello(); // "안녕 나는 undefined" (this를 user로 바인딩하지 않음)
```

#### ◆ 실습문제

1. 두 수를 더한 결과를 반환하는 add 함수 만들기
2. 세 수의 평균을 구하는 함수 (함수 표현식으로 작성)
3. 매개변수로 이름을 받아 "안녕, ○○!" 출력하는 화살표 함수 작성
4. 블록 스코프와 함수 스코프 차이를 실험해보기
  - let으로 블록 안과 밖에서 같은 이름 변수 선언 후 출력 비교
  - 함수 안에서 선언한 변수가 밖에서 보이는지 확인
5. 객체 안에 sayHi 메서드를 추가하고, this를 사용해 "나는 ○○입니다." 출력

## - Day 3 -

### 13. 객체

#### ■ 객체란

- 키(key) : 값(value) 쌍을 모아 놓은 자료형
- 예 : 학생 한명의 이름/나이/전공/점수 등을 한 덩어리로 담기 좋다.

```
const student = { name : "철수", age : 20 };
```

#### ■ 프로퍼티 읽기/쓰기

- 점 표기법과 대괄호 표기법 2가지

```
console.log(student.name);      // "철수" (점 표기법)
console.log(student["age"]);    // 20      (대괄호 표기법)
```

```
student.major = "Computer";    // 추가
student["grade"] = 3.7;        // 추가
student.age = 21;              // 수정
delete student.grade;         // 삭제
```

- 언제 대괄호 표기법을 쓰나?
  - 키가 변수로 동적일 때
  - 띄어쓰기/특수문자가 있는 키일 때(권장하지 않음)

```
const key = "home-town";
const person = {};
person[key] = "Seoul"; // OK
// person.home-town = ... // - (연산자로 해석되어 에러)
```

#### ■ 계산된 프로퍼티 이름

```
const field = "score";
const s = { name : "영희", [field] : 95 }; // { name : "영희", score : 95 }
```

#### ■ 단축 프로퍼티

- 변수명과 키 이름이 같다면 축약 가능

```
const name = "민수";
const age = 23;
const user = { name, age }; // { name : "민수", age : 23 }
```

## ■ 객체 검사 & 안전 접근

```
console.log("name" in user);           // true/false
console.log(Object.hasOwnProperty(user, "age")); // true/false (권장)

// 안전 접근(옵셔널 체이닝)
const city = user.address?.city;      // address가 없으면 undefined 반환, 예러 X

// 기본값(널 병합)
const nickname = user.nickname ?? "손님"; // null/undefined면 "손님"
```

## ■ 얕은 복사 vs 깊은 복사

```
const a = { nested: { x: 1 } };

// 얕은 복사(1단계만 복사)
const b = { ...a };
b.nested.x = 999;
console.log(a.nested.x); // 999 (같은 참조라 함께 변함)

// 깊은 복사(전부 새로 복사)
const c = structuredClone(a); // 최신 브라우저/Node에서 지원
c.nested.x = 123;
console.log(a.nested.x); // 999 (영향 없음)

// 함수/Date/LargeInt 등 손실
const d = JSON.parse(JSON.stringify(a));
```

## 14. 배열

- 배열은 여러개의 데이터를 한 줄로 차곡차곡 담아놓는 상자이다.
- 자바스크립트에서 배열은 Array 객체로, 순서가 있는 컬렉션을 표현한다.
- 여러값을 하나의 변수로 묶어서 다룰 수 있게 해주는 자료 구조이다.
- 컬렉션이란 여러개의 항목들을 모아놓은 것, 배열은 항목의 순서(Index)을 기억하고, 인덱스를 통해 항목에 접근할 수 있습니다.
- 배열 생성
  - 리터럴 표기법

```
const fruits = ["사과", "바나나", "포도"];
```

- 생성자 방식

```
const arr = new Array("사과", "바나나", "포도");
```

- length 속성

- 배열의 항목 개수를 저장하고 있는 속성
- 배열에 항목을 추가/제거하면 자동으로 변함

```
console.log(fruits.length); // 3
```

- 인덱스(Index)와 요소(Element)

- 배열 요소는 0,1,2,... 순서 있는 번호(인덱스)로 식별됨. 첫 번째 요소 인덱스는 0.
- 만약 잘못된 인덱스를 쓰면 undefined가 나옴

```
console.log(fruits[5]); // undefined
```

- 자주 쓰이는 변경 메서드와 비변경 메서드

- 변경 메서드 : push, pop, shift, unshift, splice, sort, reverse
- 비변경 메서드 : slice, map, filter, concat, toSorted, toReversed

```
// 변경 예시
```

```
const arr = [1, 2, 3];
arr.push(4);           // [1,2,3,4] (원본 변경)
arr.splice(1, 1, 99); // [1,99,3,4]
```

```
// 비변경 예시
```

```
const arr2 = [1, 2, 3];
const arr3 = arr2.slice(0, 2); // [1,2], arr2 그대로
const arr4 = arr2.map(n => n*2); // [2,4,6], arr2 그대로
```

- 검색 / 탐색

```
const nums = [5, 10, 15, 20];
nums.includes(10);          // true
nums.indexOf(15);          // 2
nums.find(n => n > 12);    // 15 (조건 만족하는 첫 값)
nums.findIndex(n => n > 12); // 2
```

- 정렬

```
const n = [10, 2, 5];
n.sort();           // ["10","2","5"] 문자열 기준 → [10, 2, 5] 그대로처럼 보일 수
있음
n.sort((a, b) => a - b); // 숫자 오름차순 → [2,5,10]

// 원본 보존하고 싶다면 (최신)
const sorted = n.toSorted((a, b) => a - b);
```

### ■ 배열 합치기 / 복사

```
const a1 = [1, 2];
const a2 = [3, 4];
const merged = a1.concat(a2); // [1,2,3,4] (비파괴)
const merged2 = [...a1, ...a2]; // 스프레드

const copy = [...a1];          // 얇은 복사
```

### ■ 배열의 장점

- 여러 값을 하나의 변수로 관리 → 코드 간결화
  - 반복문을 통한 일괄 처리 가능
  - 인덱스를 통한 빠른 접근
  - 다양한 내장메서드로 가공이 쉬움
- ### ■ 배열의 한계
- 중간 삽입/삭제가 느릴 수 있음
  - 배열 앞에서 요소를 삭제하면, 뒤의 요소들을 모두 한 칸씩 당겨야 함
  - 크기가 동적으로 늘어나지만, 메모리 관리 측면에서는 비효율적일 수 있음
  - 대량의 데이터에 특정 조건 탐색이 많을 경우, 다른 자료구조가 더 적합

## 15. 구조분해 할당 & 스프레드

### ■ 구조분해

- 배열이나 객체에서 값을 꺼내 변수에 간단히 담을 수 있는 문법
- 구조를 분해해서 변수에 할당한다는 의미

### ■ 객체 구조 분해

```
const student = { name: "철수", age: 20, major: "CS" };
const { name, age } = student;           // 같은 변수명으로 꺼내기
const { major: m = "미정" } = student; // 이름 바꾸기 + 기본값
```

### ■ 배열 구조분해

```
const fruits = ["사과", "바나나", "포도"];
const [first, second] = fruits;           // "사과", "바나나"
const [head, , tail] = fruits;          // "사과", "포도" (중간 생략)
const [x, y = "기본"] = ["값"];         // x="값", y="기본"
```

### ■ 함수 매개변수 구조분해

```
function printUser({ name, age }) {
  console.log(`${name}(${age})`);
}

printUser({ name: "영희", age: 21 });
```

### ■ 스프레드 문법

- ... 점 세 개를 사용
  - 배열이나 객체를 펼친다는 의미
- ### ■ 스프레드 vs 레스트
- 스프레드 : 펼치기 → 복사/합치기
  - 레스트 : 나머지 모으기 → 파라미터/구조분해에서 잔여값 모으기

```
// 스프레드
const base = { a: 1, b: 2 };
const ext = { ...base, b: 99, c: 3 }; // { a:1, b:99, c:3 }

// 레스트(객체)
const user = { id: 1, name: "철수", age: 20 };
const { id, ...profile } = user; // id=1, profile={ name:"철수", age:20 }

// 레스트(함수 매개변수)
function sum(...nums) {
  return nums.reduce((a, c) => a + c, 0);
}
sum(1,2,3,4); // 10
```

## 16. Tip

- sort()는 원본 배열을 바꾼다 → 원본 보존하고 싶으면 toSorted()(최신) 또는 slice().sort()로 복사 후 정렬.
- 스프레드는 얕은 복사라서 중첩된 값은 같이 변gka → 깊은 복사 필요 시 structuredClone 사용.
- for...in은 객체 키 순회, 배열에는 for...of나 forEach/map 사용을 권장.
- 없는 경로 접근 시 obj.a.b는 에러 → obj.a?.b처럼 옵셔널 체이닝으로 안전하게.
- null과 undefined에 기본값 주려면 ??(널 병합) 사용. (||는 0/빈문자열도 거짓으로 봄)

## - Day 4 -

### 17. 조건문

- 조건문은 프로그램이 상황에 따라 다른 동작을 하도록 분기 시키는 구문
- if 문
  - if : 조건이 맞으면 실행
  - else if : 앞 조건이 틀리면 다른 조건 검사
  - else : 위 조건이 모두 틀리면 실행

```
let score = 85;

if (score >= 90) {
    console.log("A 학점");
} else if (score >= 80) {
    console.log("B 학점");
} else {
    console.log("C 학점 이하");
}
```

- 만약(if) ~ 라면 .. 하고, 아니면 .. 한다 라는 뜻
- 우리가 일상에서 쓰는 말 그대로
  - 만약 비가 온다면 우산을 챙긴다.
  - 그렇지 않으면 우산을 안챙긴다.

```
let isRaining = true;

if (isRaining) {
    console.log("우산을 챙긴다 ");
} else {
    console.log("우산 필요 없음 ");
}
```

조건 검사 → true → 실행1  
→ false → 실행2

- pc방 요금제 : 만약(if) 돈이 1000원 이상이면 → 이용가능, 그렇지 않으면 잔액 부족
- switch 문
- 여러 경우의 수를 처리할 때 깔끔함
- break를 안 쓰면 뒤까지 실행되는 fall-through 현상이 생김(실무 버그 원인)

```
let fruit = "사과";

switch (fruit) {
  case "사과":
    console.log("□ 사과입니다");
    break;
  case "바나나":
    console.log("□ 바나나입니다");
    break;
  default:
    console.log("알 수 없는 과일");
}
```

## 18. 반복문

- 반복문은 코드 블록을 여러번 반복 실행하는 구조
- for문
  - 초기값 let i=0
  - 조건식 검사  $i < 5 \rightarrow \text{false}$ 면 종료
  - {} 블록 코드 실행
  - 증감식 i++
  - 조건식 검사

```
for (let i = 0; i < 5; i++) {
  console.log(i);
}
```

- while문
- 조건이 true인 동안 반복

```
let n = 0;
while (n < 3) {
  console.log(n);
  n++;
}
```

- doe ~ while 문
- 무조건 한 번은 실행 후 조건 검사

```
let num = 0;
do {
  console.log(num);
  num++;
} while (num < 3);
```

■ for...of

- 배열 순회

```
const fruits = ["사과", "바나나", "포도"];
for (let f of fruits) {
  console.log(f);
}
```

■ for...in

- 객체속성을 순회

```
const student = { name: "철수", age: 20 };
for (let key in student) {
  console.log(key, student[key]);
}
```

## 19. 배열 고차함수

- 고차함수(Higher-Order Function) = 다른 함수를 인자로 받거나 결과로 반환하는 함수
- 배열에서 자주쓰이는 대표 고차함수 : forEach, map, filter, reduce

■ forEach (단순반복)

- 반환값 없음, 단순히 모든 요소에 대해 실행

```
const numbers = [1, 2, 3];
numbers.forEach(n => {
  console.log(n * 2);
});
```

■ map (새 배열 만들기)

- 원본배열을 변형하지 않고 새 배열 반환

```
const numbers = [1, 2, 3];
const doubled = numbers.map(n => n * 2);
console.log(doubled); // [2, 4, 6]
```

■ filter (조건으로 거르기)

- 조건을 만족하는 요소만 모아 새 배열 생성
- 검색, 조건 필터링 구현에 유용

```
const scores = [45, 80, 90, 30];
const passed = scores.filter(s => s >= 60);
console.log(passed); // [80, 90]
```

■ reduce (누적 계산)

■ 배열 전체를 하나의 값으로 줄임

- acc : 누적 값
  - cur : 현재 요소
  - 0 : 초기값
- 합계, 평균, 그룹화 등 집계 연산에 필수

```
const numbers = [1, 2, 3, 4];
const sum = numbers.reduce((acc, cur) => acc + cur, 0);
console.log(sum); // 10
```

■ find , findIndex(찾기)

■ 조건을 만족하는 첫 번째 요소 또는 인덱스 찾기

```
const users = [{id:1,name:"철수"},{id:2,name:"영희"}];
const user = users.find(u => u.id === 2);
console.log(user); // {id:2, name:"영희"}
```

- Day 5 -

오늘 할일 미니 프로젝트

## - Day 6 -

### 1. DOM 이란 무엇인가?

- DOM (Document Object Model) = 웹 페이지를 자바스크립트가 다룰 수 있도록 구조화한 “트리(tree) 형태의 모델”
- HTML 요소 하나하나가 노드(node)로 표현됨.
- JS에서는 DOM을 통해 요소를 찾고(선택), 바꾸고(조작), 추가/삭제할 수 있음.
- 쉽게 말하면

웹 브라우저는 집 설계도(HTML)를 보고 실제 집(DOM)을 지어둡니다.

자바스크립트는 이 집에 들어가서 “불 켜기/끄기, 가구 옮기기, 벽 칠하기” 같은 일을 할 수 있는 주인공이에요.

### 2. DOM 요소선택 ( Element Selection )

- DOM에 있는 HTML 요소를 먼저 찾아야 조작할 수 있습니다.

#### 가) 기본 선택

```
document.getElementById("title");           // id로 선택  
document.getElementsByClassName("item"); // class로 선택  
document.getElementsByTagName("p");      // 태그 이름으로 선택
```

#### 나) css 선택자 사용 ( 추천 방식 )

```
document.querySelector("#title"); // id 선택  
document.querySelector(".item"); // class 선택  
document.querySelectorAll("p"); // 모든 &p& 선택
```

- querySelector / querySelectorAll은 CSS 선택자를 그대로 쓸 수 있어서 React나 현대 개발에서 가장 많이 쓰임.
- querySelector는 CSS 선택자와 똑같이 쓰기 때문에 배우기 쉽습니다.
- Tip: 실무에서도 거의 무조건 querySelector 씁니다.

### 3. DOM 요소 조작 ( Manipulation )

#### 가) 텍스트/HTML 변경

```
let title = document.querySelector("#title");  
title.innerText = "새로운 제목"; // 순수 텍스트만 변경  
title.innerHTML = "&b&굵은 제목&/b&"; // HTML 태그 포함 가능
```

#### 나) 속성 변경

```
let img = document.querySelector("img");

img.setAttribute("src", "dog.png");
console.log(img.getAttribute("src"));
```

#### 다) 스타일 변경

```
let box = document.querySelector(".box");

box.style.color = "red";
box.style.backgroundColor = "yellow";
```

- 실무에서는 CSS 클래스(classList.add/remove/toggle)를 조작하는 방법을 더 권장

```
box.classList.add("active");
box.classList.remove("hidden");
box.classList.toggle("dark-mode");
```

## 4. 이벤트와 이벤트 핸들링

#### 가) 이벤트란?

- **이벤트(Event)** = 사용자가 하는 행동
- 클릭, 입력, 스크롤, 마우스 올리기, 키보드 누르기 등
- **이벤트 핸들링(Event Handling)** = “어떤 일이 일어났을 때 실행할 코드”를 연결하는 것

#### 나) 이벤트 등록

```
let btn = document.querySelector("#myBtn");

btn.addEventListener("click", function() {
    alert("버튼이 눌렸습니다!");
});
```

#### 다) 실무에서는 화살표 함수도 자주 사용

```
btn.addEventListener("click", () => {
    console.log("버튼 클릭됨");
});
```

#### 라) 대표적인 이벤트 종류

- click : 버튼 클릭
- input : 입력창에 글자 입력
- change : 선택 값 변경 (예: 드롭다운)
- keydown : 키보드 입력
- submit : 폼 제출

#### 마) 이벤트 객체 (event object)

바) 이벤트가 발생하면, 브라우저는 자동으로 이벤트 정보를 담은 객체를 전달함.

```
document.querySelector("#myInput")
    .addEventListener("keydown", function(e) {
        console.log(e.key); // 어떤 키 눌렀는지 출력
    });
}
```

## 5. 생활 속 비유

### ■ DOM = 아파트

- 각 집(요소)을 찾아서, 페인트칠(스타일), 간판 바꾸기(속성), 글자 갈아끼우기(innerText) 할 수 있음

### ■ 이벤트 = 초인종

- 누군가 초인종을 누르면(이벤트 발생) → 주인이 나와서 행동(함수 실행)

### ■ 이벤트 핸들링 = 자동화 장치

- “초인종을 누르면 자동으로 문 열어주기”처럼, 조건에 따라 자동 실행되는 코드

## 6. 알아두면 좋은 Tip

#### 가) DOM은 그림으로 이해

- HTML 태그 하나하나를 “박스”라고 생각하고, 트리처럼 이어져 있다고 상상

#### 나) querySelector는 CSS랑 똑같다고 기억

- #id, .class, tag → 웹디자인 수업에서 CSS를 했던 사람은 금방 적응함

#### 다) innerText vs innerHTML 구분

- innerText = 글자만

- innerHTML = 태그까지 포함

#### 라) 이벤트 핸들링은 ‘조건’이 아니라 ‘반응’

- if문은 “조건에 따라 실행”

- 이벤트는 “사용자가 행동할 때 실행”

#### 마) 하나씩 테스트

- DOM 조작이나 이벤트는 오타 하나로 안 돌아감

- console.log()로 단계마다 확인하는 습관이 중요

## 7. 실습 문제

### ■ 문제 1.

- 버튼 클릭 시 배경색이 바뀌는 박스 만들기

- 입력창에 이름을 입력하면 안녕하세요, \_\_\_님! 출력하기

- 문제 2.
  - +1, -1 버튼으로 카운터 만들기
  - 체크박스 클릭 시 텍스트에 취소선 그어지게 하기

## 8. 정리

- DOM = HTML을 코드로 다룰 수 있게 한 구조
- 요소 선택 : querySelector / querySelectorAll
- 요소 조작 : .innerText, .innerHTML, .style, .classList
- 이벤트 : 사용자의 행동, addEventListener로 연결
- 실습을 통해 “내가 만든 웹이 살아 움직이는 경험”을 하는 것이 중요

## - Day 7 -

### 1. localStorage

#### ■ localStorage 개념

브라우저에 데이터를 저장할 수 있는 공간

용량 : 약 5MB

특징 : 브라우저를 껐다 켜도 데이터가 남아 있음 (세션과 다름)

간단히 말해, localStorage는 브라우저 안의 작은 메모장

#### ■ 사용처

UI 환경설정 저장 : 다크모드, 언어, 글자 크기 등

최근 본 항목/검색어 기록

간단한 임시 저장 : 품 작성 중인 내용 임시 보관(초안/드래프트)

가벼운 캐시 : 작은 API 응답을 잠깐 저장(시간 제한 두기)

```
// 저장  
localStorage.setItem("key", "value");  
  
// 불러오기  
let value = localStorage.getItem("key");  
console.log(value);  
  
// 삭제  
localStorage.removeItem("key");  
  
// 전체 삭제  
localStorage.clear();
```

#### ■ JSON과 함께 쓰기

■ localStorage는 문자열만 저장 가능 → 객체/배열은 JSON 변환 필요

```
let user = { name: "홍길동", age: 25 };  
  
// 저장  
localStorage.setItem("user", JSON.stringify(user));  
  
// 불러오기  
let savedUser = JSON.parse(localStorage.getItem("user"));  
console.log(savedUser.name); // "홍길동"
```

## Tip

1. 동기(synchronous) API라서, 큰 데이터를 찾게 저장하면 UI가 잠깐 멈출 수 있어요
  - → 데이터가 크거나 자주 쓰면 IndexedDB(비동기) 고려
2. 보안
  - XSS가 발생하면 localStorage 내용이 유출될 수 있어요. 민감 정보(토큰/개인정보) 저장 금지
  - 인증 토큰은 가능한 HTTP-Only 쿠키나 메모리 사용 등 대안 검토
3. 문자열만 저장 → JSON.stringify/parse는 try/catch로 감싸 오류 대비(사용자가 개발자 도구로 값을 바꾸거나, 손상될 수 있음)
4. 네이밍/버저닝
  - 키 앞에 접두어 붙이기 : "myapp:v1:theme", "myapp:v1:user"
  - 스키마가 바뀌면 버전 올리고 마이그레이션 로직 추가
5. TTL(만료시간) 패턴
  - 저장할 때 시간도 같이 넣고, 꺼낼 때 만료 확인

```
function setWithTTL(key, value, ttlMs) {
  const record = { value, expiresAt: Date.now() + ttlMs };
  localStorage.setItem(key, JSON.stringify(record));
}

function getWithTTL(key) {
  try {
    const raw = localStorage.getItem(key);
    if (!raw) return null;
    const { value, expiresAt } = JSON.parse(raw);
    if (Date.now() > expiresAt) { localStorage.removeItem(key); return null; }
    return value;
  } catch { return null; }
}
```

## 2. fetch API

### fetch 개념

서버(또는 외부 API)에 요청(request)을 보내고 응답(response)을 받는 함수

HTTP 요청을 쉽게 할 수 있음

□ 쉽게 말해, fetch는 웹에서 데이터 가져오기 배달 서비스

공공/외부 API에서 데이터 가져오기(날씨/영화/뉴스 등)

백엔드 서버와 통신(상품목록, 로그인, 글쓰기 등)

### 2.3 fetch 동작 흐름

fetch(URL) → 요청 보내기

서버가 응답 → response 객체  
response.json()으로 JSON 변환  
변환된 데이터를 화면에 표시

---

## 2.4 실습

### 공공 API 불러오기

```
fetch("https://jsonplaceholder.typicode.com/todos/1")
  .then(res => res.json())
  .then(data => {
    console.log("제목:", data.title);
  });

```

---

## 3. 미니 프로젝트: 메모장 저장 & API 데이터 출력

### 목표

입력창에 메모 작성 → localStorage에 저장  
새로고침해도 메모 유지  
버튼 클릭 시 API에서 데이터 불러와 화면에 표시

---

### 코드 예시

```
<input type="text" id="memoInput" placeholder="메모를 입력하세요">
<button id="saveBtn">저장</button>
<p id="memoDisplay"></p>

<button id="apiBtn">API 불러오기</button>
<p id="apiResult"></p>

<script>
  const input = document.querySelector("#memoInput");
  const saveBtn = document.querySelector("#saveBtn");
  const display = document.querySelector("#memoDisplay");

  // 저장된 메모 불러오기
  display.innerText = localStorage.getItem("memo") || "";

  // 메모 저장
  saveBtn.addEventListener("click", () => {
    localStorage.setItem("memo", input.value);
  });

  // API 불러오기
  apiBtn.addEventListener("click", () => {
    fetch("https://jsonplaceholder.typicode.com/todos/1")
      .then(res => res.json())
      .then(data => {
        memoDisplay.textContent = data.title;
      });
  });
</script>
```

```
saveBtn.addEventListener("click", () =& {
    localStorage.setItem("memo", input.value);
    display.innerText = input.value;
});

// API 데이터 불러오기
document.querySelector("#apiBtn").addEventListener("click", () =& {
    fetch("https://jsonplaceholder.typicode.com/todos/1")
        .then(res =& res.json())
        .then(data =& {
            document.querySelector("#apiResult").innerText = data.title;
        });
});
</script>
```

---

### ✓ Tip

localStorage는 데이터베이스(DB)의 아주 간단한 버전이라고 생각하면 이해 쉬움  
fetch는 “외부에서 JSON을 가져와 DOM에 뿐린다”로 요약  
console.log()로 데이터가 제대로 들어왔는지 항상 확인해보기

---

## 4. 오늘 배운 핵심

localStorage = 브라우저 저장소 (데이터 유지 가능)

fetch API = 외부 데이터 불러오기

□ 내 웹사이트가 기억하고, 외부와 소통하는 첫걸음!

---

## 5. 도전 과제

오늘 할 일(Todo)을 localStorage에 저장해서, 새로고침해도 유지되도록 만들기

fetch API로 영화 검색 API(OMDb, TMDB 등) 불러와서 영화 제목 목록 출력하기

## - Dya 8 -

# 1. JSON (JavaScript Object Notation)

JSON이란?

데이터를 주고받기 위한 표준 형식

자바스크립트의 객체 문법을 차용 → 다른 언어에서도 사용 가능

웹 API, 서버/클라이언트 간 데이터 교환에 가장 많이 쓰임

□ 쉽게 말해 : JSON은 데이터를 담은 택배 상자, 누구나 열 수 있고, 어디로든 보낼 수 있음

---

JSON 형식

Key-Value 쌍 (키는 반드시 문자열 " "로 감싸야 함)

숫자, 문자열, 불리언, 배열, 객체 포함 가능

```
{  
  "name": "홍길동",  
  "age": 25,  
  "isStudent": false,  
  "hobbies": ["독서", "게임"],  
  "address": { "city": "서울", "zip": "12345" }  
}
```

---

JSON과 자바스크립트 객체 차이

// JS 객체

```
let obj = { name: "홍길동", age: 25 };
```

// JSON (문자열)

```
let json = '{"name":"홍길동","age":25}';
```

---

변환 방법

// 객체 → JSON 문자열

```
let jsonStr = JSON.stringify(obj);
```

// JSON 문자열 → 객체

```
let parsedObj = JSON.parse(jsonStr);
```

## JSON 실습

user 객체 만들고 localStorage에 JSON으로 저장

저장된 JSON 문자열을 꺼내서 다시 객체로 변환

이름과 나이를 출력해보기

---

## 2. 비동기 처리 (async/await)

동기 vs 비동기

**동기(Synchronous)** : 작업이 순서대로, 하나 끝나야 다음 실행

**비동기(Asynchronous)** : 기다리는 동안 다른 작업도 진행 가능

□ 예시

동기 = 편의점에서 계산대 1개, 손님이 계산 끝날 때까지 다음 사람 기다림

비동기 = 무인 계산대 여러 개, 계산 중에도 다른 사람 처리 가능

---

### fetch + 비동기 문제

```
let data = fetch("https://jsonplaceholder.typicode.com/todos/1");
console.log(data); // Promise { &pending& }
```

□ fetch 결과는 바로 값이 아니라 **Promise**(약속).

즉, “나중에 결과 줄게”라는 종이 쪽지 같은 것.

---

### async/await 기본 문법

```
async function getTodo() {
  try {
    let response = await fetch("https://jsonplaceholder.typicode.com/todos/1");
    if (!response.ok) throw new Error("HTTP 오류: " + response.status);
    let data = await response.json();
    console.log(data);
  } catch (err) {
    console.error("에러 발생:", err);
  }
}
getTodo();
```

async : 함수가 비동기 함수를 쓴다는 표시

await : Promise가 끝날 때까지 기다림

try/catch : 에러를 안전하게 처리

---

## async/await 실습

버튼 클릭 시 API 요청 → JSON 데이터 화면에 표시

로딩 중일 때 “로딩 중...” 표시

에러 발생 시 “데이터 불러오기 실패” 표시

---

## Tip

### JSON

JSON은 반드시 " " 큰따옴표 사용 (작은따옴표 불가)

undefined, function은 JSON에 저장할 수 없음

JSON.parse 시 에러 발생 가능 → try/catch로 감싸기

## async/await

await는 반드시 async 함수 안에서만 사용 가능

fetch는 404/500 에러도 Promise 성공으로 취급 → res.ok 확인 필수

네트워크 불안정 고려해서 항상 에러 처리(catch or try/catch)

---

## 3. 미니 프로젝트: 영화 검색 앱 (간단 버전)

### 요구사항

입력창에 영화 제목 입력

버튼 누르면 API 요청

JSON 데이터에서 영화 목록 가져와 화면에 출력

<input id="query" placeholder="영화 제목 입력">

<button id="searchBtn">검색</button>

<ul id="results"></ul>

```
async function searchMovie() {  
  const query = document.querySelector("#query").value;  
  const results = document.querySelector("#results");  
  results.innerHTML = "로딩 중...";
```

```
try {
    const res = await fetch(`https://www.omdbapi.com/?apikey=demo&s=${query}`);
    if (!res.ok) throw new Error("HTTP 에러 " + res.status);

    const data = await res.json();
    results.innerHTML = "";

    if (data.Search) {
        data.Search.forEach(movie => {
            const li = document.createElement("li");
            li.textContent = movie.Title + " (" + movie.Year + ")";
            results.appendChild(li);
        });
    } else {
        results.innerHTML = "검색 결과 없음";
    }
} catch (err) {
    results.innerHTML = "데이터 불러오기 실패";
    console.error(err);
}
}

document.querySelector("#searchBtn").addEventListener("click", searchMovie);
```

---

## 6. 오늘 핵심

**JSON** = 데이터 교환 표준, `stringify/parse`로 변환

**async/await** = 비동기 코드를 동기처럼 읽기 쉽게 작성

**실무 포인트** = 항상 에러 처리 + 로딩 상태 처리

## WORK BOOK

### 📚 JSON 이해하기

JSON은 데이터를 주고받기 위한 \_\_\_\_\_ 형식입니다.

자바스크립트의 객체 문법과 비슷하지만, 모든 키는 반드시 \_\_\_\_\_로 감싸야 합니다.

JSON은 서버 ↔ 브라우저 간 \_\_\_\_\_ 교환에 가장 많이 사용됩니다.

## 💡 JSON 예시

아래는 사용자 정보를 담은 JSON입니다.

```
{  
    "name": "홍길동",  
    "age": 25,  
    "isStudent": false,  
    "hobbies": ["독서", "게임"],  
    "address": { "city": "서울", "zip": "12345" }  
}
```

- JSON에서 name 키의 값은? → \_\_\_\_\_
- hobbies 배열의 두 번째 값은? → \_\_\_\_\_
- 

## 💡 JSON과 JS 객체의 차이

// 자바스크립트 객체

```
let obj = { name: "홍길동", age: 25 };
```

// JSON 문자열

```
let json = '{"name":"홍길동", "age":25}';
```

- JSON은 단순히 텍스트(문자열)이에요.
  - 자바스크립트에서 사용하려면 객체로 변환(parse) 해야 합니다.
- 

## 💡 실습 JSON 변환

```
let user = { name: "철수", age: 30 };
```

// (1) 객체 → JSON 문자열로 바꾸기

```
let jsonData = JSON._____ (user);
```

// (2) JSON 문자열 → 객체로 다시 바꾸기

```
let objData = JSON._____ (jsonData);
```

```
console.log(objData.name);
```

---

## 💡 async / await 이해하기

### □ 개념 요약

**동기(synchronous)** : 한 작업이 끝나야 다음 작업 실행

**비동기(asynchronous)** : 기다리는 동안 다른 작업도 진행 가능

async/await는 비동기 코드를 마치 순서대로 실행되는 것처럼 쉽게 읽을 수 있게 해줍니다.

---

### 비유

**동기** : 편의점에 계산대 1개 → 손님이 계산 끝날 때까지 다음 손님 대기

**비동기** : 셀프 계산대 여러 개 → 동시에 계산 가능

---

### 기본 문법

```
async function getData() {  
  try {  
    let response = await fetch("https://jsonplaceholder.typicode.com/todos/1");  
    let data = await response.json();  
    console.log(data);  
  } catch (error) {  
    console.error("에러 발생:", error);  
  }  
}
```

□ async 함수 안에서는 \_\_\_\_\_ 키워드를 써서 Promise가 끝날 때까지 기다립니다.

□ 오류가 나면 try / \_\_\_\_\_ 블록으로 예외를 처리합니다.

---

### 주의사항

|       |  |
|-------|--|
| 항목    | 주의 포인트                                 |
| await | 반드시 async 함수 안에서만 사용 가능                |
| fetch | 404/500도 에러 아님 → res.ok로 상태 체크 필요      |
| JSON  | undefined, function은 JSON에 저장 불가       |
| 보안    | fetch 시 API 키 노출 주의 (.env 파일이라도 노출 가능) |

---

## 💡 실습 영화 검색 앱 만들기

입력창에 영화 제목을 입력하면

버튼 클릭 → API 호출 (OMDb 무료 API 사용)

JSON 데이터로 영화 제목과 연도 출력

---

### 준비 코드

```
<input id="query" placeholder="영화 제목 입력">
<button id="searchBtn">검색</button>
<ul id="results"></ul>
```

---

아래 코드를 완성해보세요.

```
async function searchMovie() {
    const query = document.querySelector("#query").value;
    const results = document.querySelector("#results");
    results.innerHTML = "_____"; // 로딩 메시지

    try {
        const res = await fetch(`https://www.omdbapi.com/?apikey=demo&s=${query}`);
        if (!res.ok) throw new Error("HTTP 에러");

        const data = await res.json(); // JSON 변환
        results.innerHTML = "";

        if (data.Search) {
            data.Search.forEach(movie => {
                const li = document.createElement("li");
                li.textContent = `${movie.Title} (${movie.Year})`;
                results.appendChild(li);
            });
        } else {
            results.innerHTML = "검색 결과 없음";
        }
    } catch (err) {
        results.innerHTML = "_____ □"; // 에러 메시지
    }
}

document.querySelector("#searchBtn")
    .addEventListener("click", searchMovie);
```

## - Day 9 -

# React란 무엇인가?

### 개념 요약

React는 사용자 인터페이스(UI)를 만들기 위한 자바스크립트 라이브러리입니다.

HTML + JS + CSS로 구성된 웹을 “컴포넌트”라는 조각 단위로 관리하게 해줍니다.

페이스북(Meta)이 만들었고, 현재 전 세계에서 가장 많이 사용되는 프론트엔드 기술입니다.

### React의 특징

| 특징                  | 설명                         |
|---------------------|----------------------------|
| 컴포넌트 기반             | UI를 재사용 가능한 컴포넌트 단위로 구성    |
| 단방향 데이터 흐름          | 데이터가 부모 → 자식 방향으로 흐름       |
| 가상 DOM(Virtual DOM) | 변경된 부분만 효율적으로 업데이트         |
| 선언형 프로그래밍           | “어떻게”보다 “무엇을 보여줄지”를 코드로 표현 |

### 예시로 이해하기

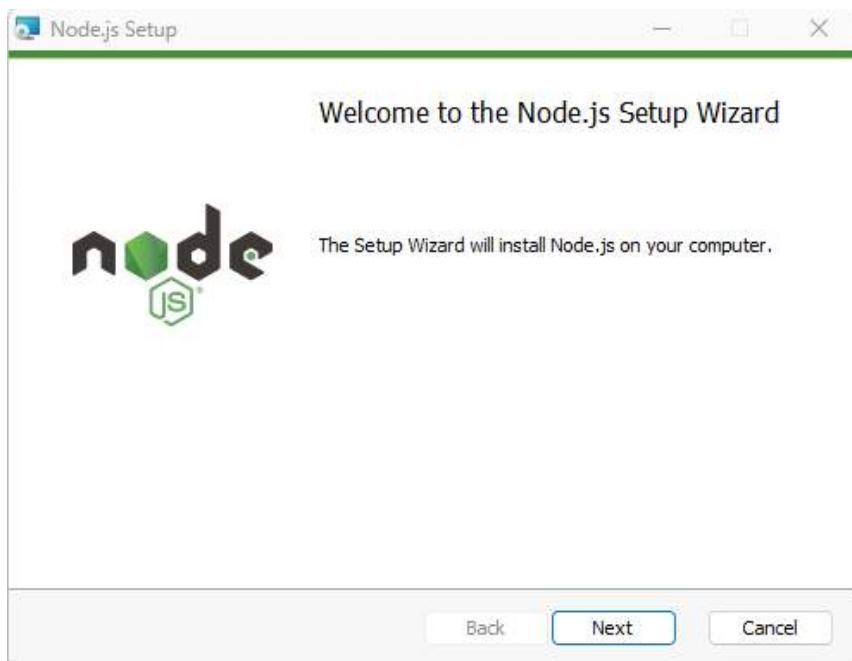
“HTML은 브라우저로 직접 그림을 그리는 것,

React는 도형 조각(컴포넌트)을 미리 만들어 조합하는 것.”

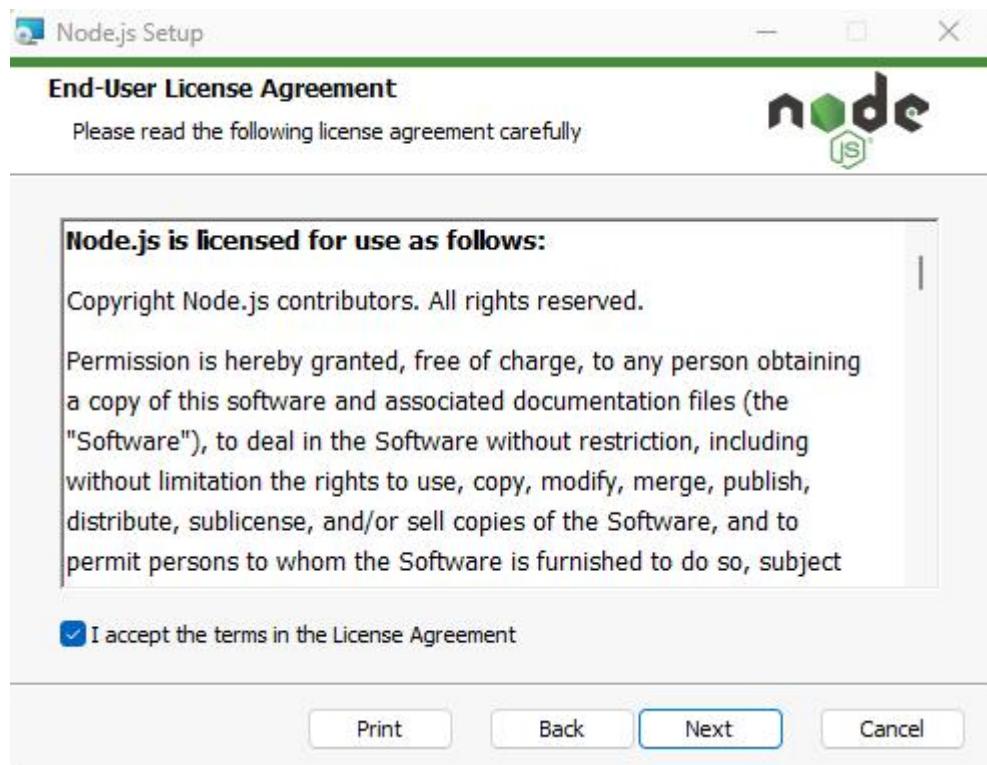
### 개발 환경 세팅 (Vite + Node.js + VS Code)

#### Node.js 설치

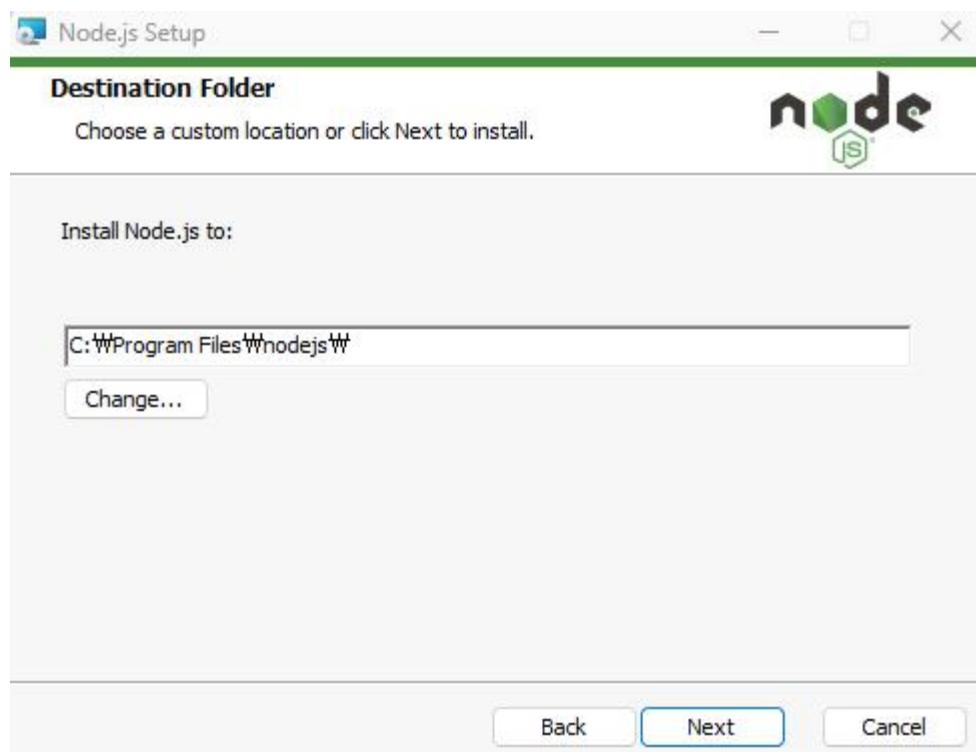
<https://nodejs.org> → LTS 버전 다운로드



next 클릭



체크박스 체크 후 next 클릭



next 클릭

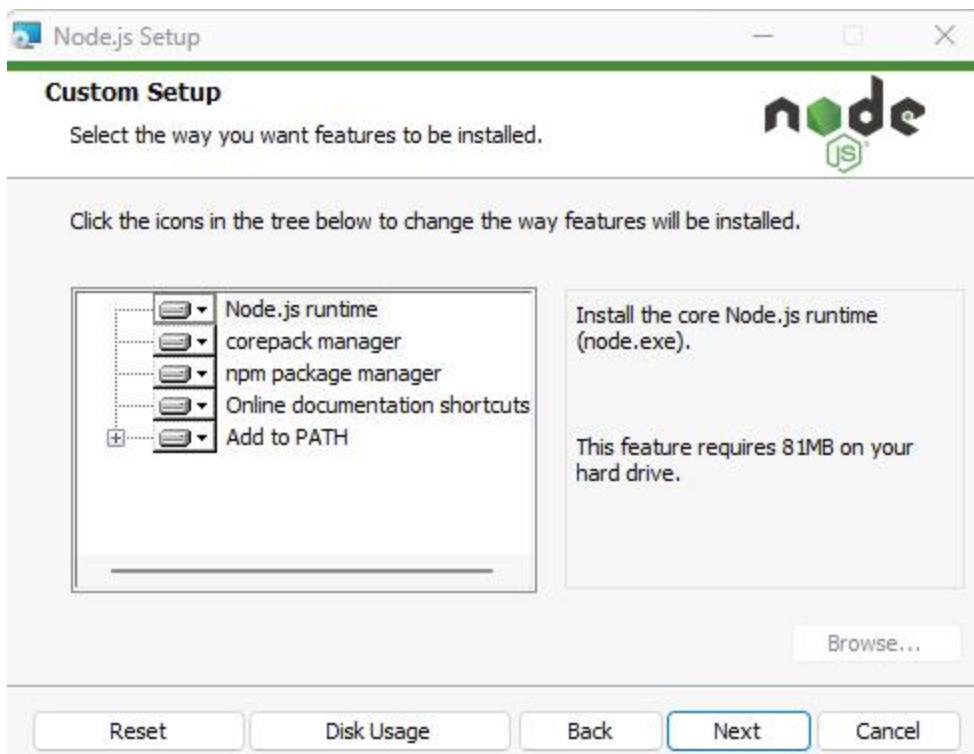


그림 34

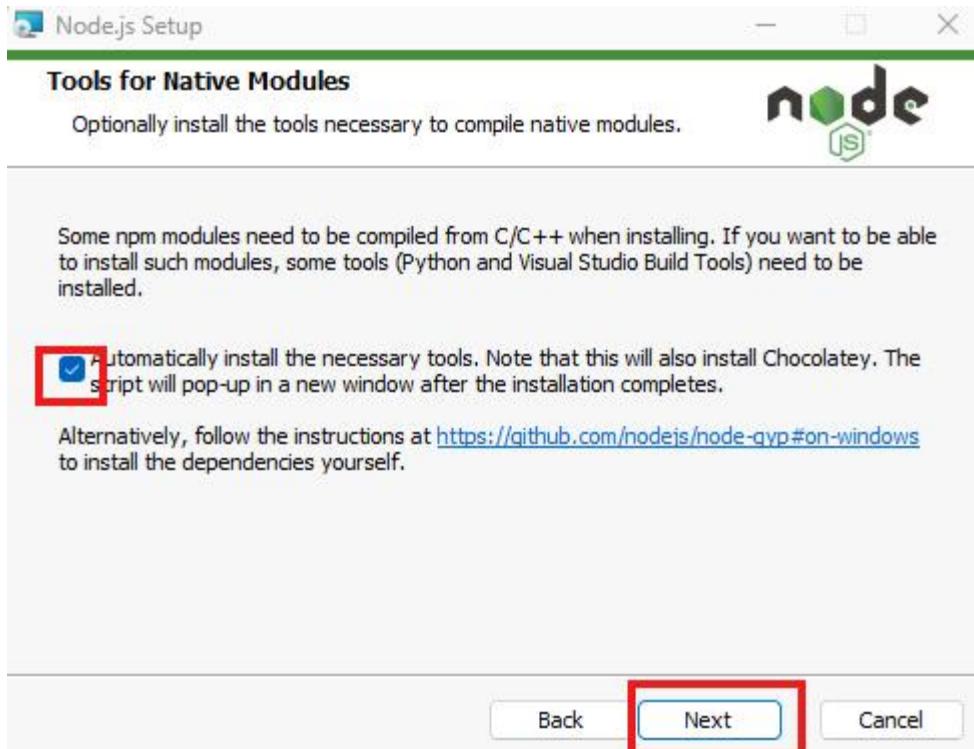


그림 35

Automatically 왼쪽 체크하고 Next 클릭

- Automatically install 체크 안 했을 경우 나중에 후회하는 경우가 많으므로 꼭 설치하는 것을 추천

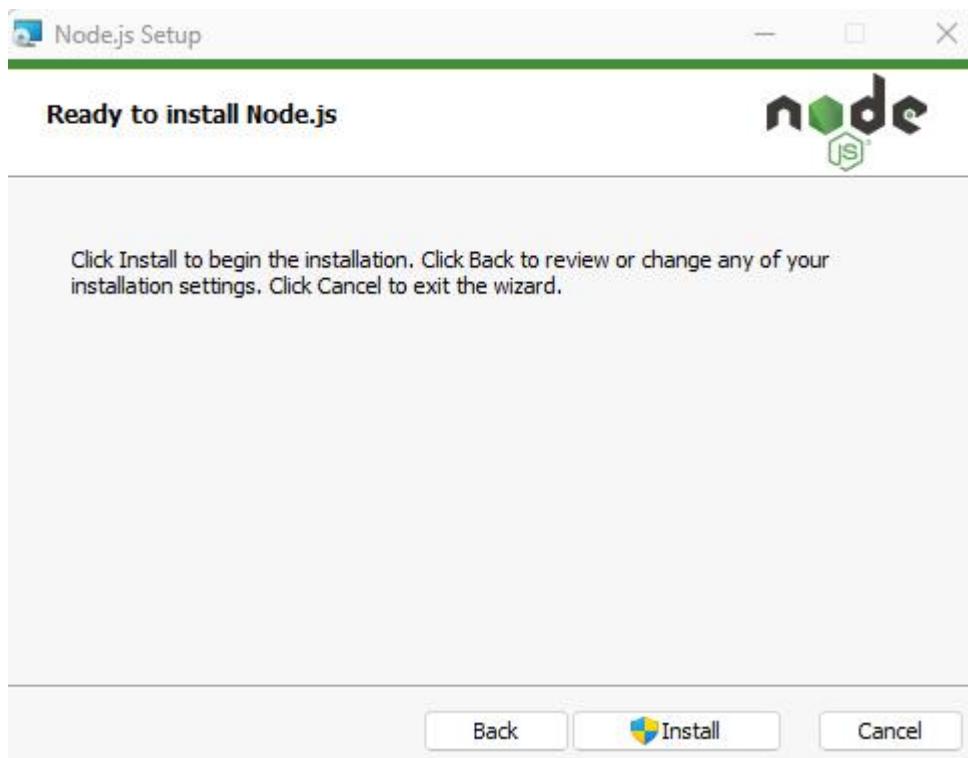


그림 36

설치 중에 중간에 팝업화면이 뜨면서 설치할 건지 물어보면 예(Yes) 누르고 넘어가기

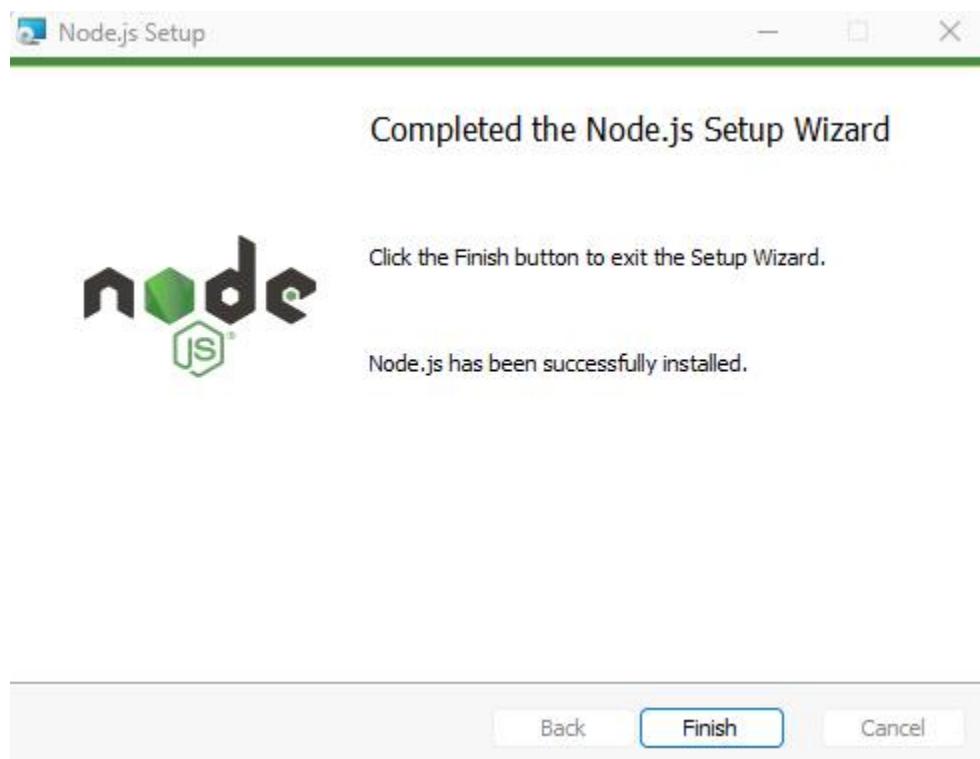


그림 37

```
Install Additional Tools for No X + ▾
=====
Tools for Node.js Native Modules Installation Script
=====

This script will install Python and the Visual Studio Build Tools, necessary
to compile Node.js native modules. Note that Chocolatey and required Windows
updates will also be installed.

This will require about 7 GiB of free disk space, plus any space necessary to
install Windows updates. This will take a while to run.

Please close all open programs for the duration of the installation. If the
installation fails, please ensure Windows is fully updated, reboot your
computer and try to run this again. This script can be found in the
Start menu under Node.js.

You can close this window to stop now. Detailed instructions to install these
tools manually are available at https://github.com/nodejs/node-gyp#on-windows

계속하려면 아무 키나 누르십시오 . . .
```

그림 38

```
Install Additional Tools for No X + ▾
=====
Using this script downloads third party software

This script will direct to Chocolatey to install packages. By using
Chocolatey to install a package, you are accepting the license for the
application, executable(s), or other artifacts delivered to your machine as a
result of a Chocolatey install. This acceptance occurs whether you know the
license terms or not. Read and understand the license terms of the packages
being installed and their dependencies prior to installation:
- https://chocolatey.org/packages/chocolatey
- https://chocolatey.org/packages/python
- https://chocolatey.org/packages/visualstudio2019-workload-vctools

This script is provided AS-IS without any warranties of any kind
=====

Chocolatey has implemented security safeguards in their process to help
protect the community from malicious or pirated software, but any use of this
script is at your own risk. Please read the Chocolatey's legal terms of use
as well as how the community repository for Chocolatey.org is maintained.

계속하려면 아무 키나 누르십시오 . . .
```

그림 39

Windows PowerShell 실행 여부 창 예 클릭

## React 프로젝트 생성

```
npm create vite@latest my-react-app
```

Framework : React

Variant : JavaScript 선택

```
cd my-react-app
```

```
npm install
```

```
npm run dev
```

브라우저에서 <http://localhost:5173> 접속 → 첫 React 화면 확인

my-react-app/

```
|__ public/          → 이미지 등 정적 파일  
|__ src/           → 코드 작성 폴더  
|   |__ App.jsx     → 메인 컴포넌트  
|   |__ main.jsx    → React 시작점  
|   |__ components/ → 컴포넌트 폴더  
|__ package.json  
└__ vite.config.js
```

## ❖ JSX 문법

JSX란?

JavaScript 안에서 HTML처럼 UI를 표현할 수 있게 하는 React 전용 문법

사실상 React.createElement() 의 간결한 표현 방식

// HTML과 비슷하지만, 실제로는 JS 코드

```
const element = <h1>안녕하세요, React!</h1>;
```

---

## JSX 문법 규칙

| 규칙                | 예시                     | 설명                   |
|-------------------|------------------------|----------------------|
| 닫는 태그 필수          | <img />                | 홀로 있는 태그도 반드시 닫기     |
| 여러 요소는 하나로 감싸기    | <div>...</div>         | 루트 태그 하나만 존재해야 함     |
| class → className | <div className="box">  | JS 예약어(class)와 충돌 방지 |
| JS 표현식은 {}        | {name}                 | 변수나 연산 결과를 표현 가능     |
| 속성값 문자열은 따옴표      |  | 속성값은 항상 문자열로 작성      |

---

## JSX 예시

```
const name = "홍길동";
const element = (
  <div>
    <h1>안녕하세요, {name}!</h1>
    <p>오늘도 즐거운 코딩 </p>
  </div>
);
export default element;
```

---

## ❖ 첫 컴포넌트 만들기

### 컴포넌트란?

화면의 일부를 구성하는 독립적인 UI 조각

함수형 컴포넌트가 주로 사용됨

```
function Welcome(props) {
  return <h1>안녕하세요, {props.name}님!</h1>;
}
```

```
export default Welcome;
```

### 사용 예시

```
import Welcome from "./components/Welcome";
```

```
function App() {
  return (
    <div>
      <Welcome name="홍길동" />
      <Welcome name="철수" />
    </div>
  );
}
```

```
export default App;
```

---

## ❖ 자주 하는 실수 & 주의 사항

| 항목        | 설명                         |
|-----------|----------------------------|
| 대문자 시작    | 컴포넌트 이름은 반드시 <b>대문자 시작</b> |
| 루트 태그 하나  | JSX는 반드시 하나의 부모 태그로 감싸야 함  |
| className | class 대신 className 사용      |
| return 팔호 | 여러 줄 JSX 반환 시 팔호로 감싸기      |
| JS 코드와 혼동 | JSX 안에서는 {} 만 JS 표현 가능     |
| 경로 오타     | import 경로 대소문자 정확히 확인      |

---

## ❖ 실습 : 나만의 명함 카드 만들기

### 목표

이름, 직업, 이메일을 JSX로 표현

CSS 클래스 적용 (className 사용)

```
function MyCard() {
  return (
    <div className="card">
      <h2>전계림</h2>
      <p>프론트엔드 개발자</p>
      <p>Email : example@naver.com</p>
    </div>
  );
}

export default MyCard;
```

확장 : 사진, 소셜 링크, 배경색 변경 기능 추가

---

## 복습 퀴즈

| 번호 | 문제   | 답 |
|----|--|---|
| 1  | React는 무엇을 만들기 위한 라이브러리인가요?                |   |
| 2  | JSX는 _____ 안에서 _____ 를 표현할 수 있는 문법입니다.     |   |
| 3  | JSX에서 class 대신 사용하는 속성은?                   |   |
| 4  | 여러 요소를 JSX로 반환할 때 반드시 하나의 _____ 로 감싸야 합니다. |   |
| 5  | 컴포넌트 이름은 반드시 _____ 로 시작해야 합니다.             |   |

## 오늘의 도전 과제

ProfileCard 컴포넌트 만들기

props : name, job, email

카드 스타일 꾸미기 (CSS className 사용)

App.jsx 에 여러 명의 프로필 카드 출력하기

map() 을 이용해 반복 렌더링 시도하기

---

## 오늘의 핵심 정리

React는 컴포넌트 기반 UI 라이브러리이다.

JSX는 HTML + JS 의 결합 문법으로, **className / {} / 루트 태그** 규칙이 중요하다.

모든 React 앱은 작은 컴포넌트의 조합으로 이루어진다.

# REACT 교육 일정 재구성

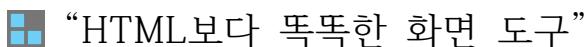
| 단계                    | 일자  | 오전  | 오후   | 학습 목표                   |
|-----------------------|-----|---|--|-------------------------|
| React<br>기초 다<br>지기   | 1일  | JSX 복습, 컴포넌트 구조 이해<br>import / export / return 개념             | props, state 실습 (카운터, 입력값 제어)                    | 컴포넌트 구조와 상태 변화 이해       |
|                       | 2일  | useState 응용 : 입력값·리스트 관리                                      | 부모→자식 데이터 전달(props) / 이벤트 핸들링(onClick, onChange) | React 데이터 흐름(상향·하향) 익히기 |
|                       | 3일  | useEffect 기초 : 렌더링 타이밍과 생명주기                                  | fetch()로 JSON 불러오기 실습 (공공데이터 예시)                 | 외부 데이터 가져오기 이해          |
|                       | 4일  | map(), 조건부 렌더링, key 개념  | 미니 프로젝트 : “날씨/환율 리스트 출력 앱”                       | 데이터 출력과 렌더링 완성          |
| Chart.js<br>시각화<br>집중 | 5일  | Chart.js & react-chartjs-2 설치 / 구조 설명                         | 하드코딩 데이터로 막대, 선 그래프 실습                           | 그래프 생성 구조 이해            |
|                       | 6일  | props 기반 Chart 데이터 변경 실습                                      | fetch() 데이터 → Chart.js 시각화                       | API 데이터 그래프화            |
|                       | 7일  | Line + Bar + Pie 등 복합 차트 구성                                   | 미니 프로젝트 : “나의 데이터 다시 보드” 제작                      | 데이터 시각화 완성              |
| Firebas<br>e 연동       | 8일  | Firebase 프로젝트 생성 / 연결   | Firestore CRUD (데이터 추가·삭제·조회)                    | React + DB 연동           |
|                       | 9일  | Firebase Auth (로그인 / 회원가입)                                    | 로그인 사용자 전용 데이터 관리                                | 인증 + 개인화 데이터 연동         |
|                       | 10일 | Firebase Hosting 배포 / 환경변수 설정                                 | 종합 실습 : “로그인 + 차트 + 저장 앱”                        | 배포 가능한 서비스 완성           |
| 프로젝트<br>예열 및<br>정리    | 11일 | 핵심 기술 총복습(JSX → useState → fetch → Chart.js → Firebase 흐름 정리) | 팀별 미니 기획 & API 테스트 → Firebase Hosting으로 배포       | 4주 프로젝트 준비 완료           |

# Day 1 – JSX, 컴포넌트, props/state 기초

## 오늘의 목표

- ✓ 리액트로 화면을 만드는 기본 구조를 이해하고
- ✓ “컴포넌트”로 화면을 나누는 방법을 배우며
- ✓ “props”와 “state”로 데이터를 다루는 기초를 익히기

## 1. React란?



“HTML보다 똑똑한 화면 도구”

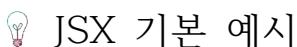
보통 HTML로 만든 웹페이지는 고정되어 있음, 버튼을 눌러도 화면이 바뀌려면 새로고침이 필요  
리액트(React)는 데이터가 바뀌면 자동으로 화면이 바뀌는 웹페이지를 만들 수 있게 해주는 도구

○ 비유

HTML은 ‘사진’, React는 ‘동영상’ 즉, 살아 있는 화면을 만드는 것

## 2. JSX문법이란?

React에서는 HTML처럼 보이는 코드를 사용하지만,  
사실은 JavaScript 안에서 화면을 만드는 문법  
이게 바로 JSX (JavaScript + XML)



JSX 기본 예시

```
// App.jsx
export default function App() {
  return (
    <div>
      <h1>안녕하세요! DW Academy입니다 </h1>
      <p>이건 JSX 문법으로 만든 화면이에요.</p>
    </div>
  );
}
```

이 코드는 HTML처럼 보이지만 실제로는 JavaScript 함수  
return 안에 들어있는 게 화면에 보여질 부분

## 💡 JSX의 중요한 규칙

| 규칙 | 설명                    | 예시   |
|----|-----------------------|--|
| 1  | 반드시 한 개의 부모 태그로 감싸야 함 | ✖ <h1></h1><p></p> → ✅ <div><h1></h1><p></p></div> |
| 2  | 태그는 닫혀야 함             | <img />, <input />                                 |
| 3  | 자바스크립트 같은 {} 로 표시     | <p>{name}</p>                                      |
| 4  | class 대신 className 사용 | <div className="box"></div>                        |

## 💻 실습 : Hello React

📁 파일 : src/App.jsx

```
export default function App() {
  const name = "홍길동";
  return (
    <div>
      <h1>안녕하세요, {name}님 </h1>
      <p>React 세계에 오신 것을 환영합니다!</p>
    </div>
  );
}
```

## 💡 예상화면

안녕하세요, 홍길동님

React 세계에 오신 것을 환영합니다!

## 3. 컴포넌트 (Component)

### UIScreen을 나누는 작은 블록

React에서는 화면 전체를 한 번에 만들지 않고, Header, Main, Footer 처럼 조각 단위(컴포넌트)로 나눈다.

#### ○ 비유

웹사이트는 레고 블록처럼 컴포넌트를 조합해서 만든 집 이다.

## 💻 실습 : 컴포넌트 만들기

📁 파일 구조

src/

- └ App.jsx
- └ Header.jsx
- └ Footer.jsx

#### 📄 Header.jsx

```
export default function Header() {  
    return <h1>DW Academy React 수업</h1>;  
}
```

#### 📄 Footer.jsx

```
export default function Footer() {  
    return <p>2025 DW Academy All Rights Reserved.</p>;  
}
```

#### 📄 App.jsx

```
import Header from "./Header";  
import Footer from "./Footer";  
  
export default function App() {  
    return (  
        <div>  
            <Header />  
            <p>이건 본문 부분입니다.</p>  
            <Footer />  
        </div>  
    );  
}
```

#### 💡 예상화면

DW Academy React 수업  
이건 본문 부분입니다.  
2025 DW Academy All Rights Reserved.

## 4. props (프로퍼티)

## “컴포넌트에 값을 전달하는 방법”

부모 컴포넌트(App)에서 자식 컴포넌트로 데이터를 넘길 때 props를 쓴다.

### 실습 : props 사용해보기

 파일 : Welcome.jsx

```
export default function Welcome(props) {
  return <h2>안녕하세요, {props.name} 님 </h2>;
}
```

 파일 : App.jsx

```
import Welcome from "./Welcome";

export default function App() {
  return (
    <div>
      <Welcome name="이순신" />
      <Welcome name="문익점" />
    </div>
  );
}
```

### 예상화면

안녕하세요, 이순신님

안녕하세요, 문익점님

○ props는 부모 → 자식으로 데이터 전달하는 통로이다.

HTML의 속성(attribute)처럼 생각하면 된다.

예: <Welcome name="이순신" />

## 5. state (상태)

### “화면에서 바뀌는 데이터”

리액트의 가장 큰 특징은 데이터가 바뀌면 화면도 자동으로 바뀐다는 것!

그 데이터를 state라고 부른다.

### 실습 : 숫자 증감 버튼 만들기

 파일 : Counter.jsx

```

import { useState } from "react";

export default function Counter() {
  const [count, setCount] = useState(0); // 기본값 0

  return (
    <div>
      <h3>현재 숫자 : {count}</h3>
      <button onClick={() => setCount(count + 1)}>+ 증가</button>
      <button onClick={() => setCount(count - 1)}>- 감소</button>
    </div>
  );
}

```

## 📁 App.jsx

```

import Counter from "./Counter";

export default function App() {
  return (
    <div>
      <h1>카운터 예제</h1>
      <Counter />
    </div>
  );
}

```

## 💡 예상화면

카운터 예제

현재 숫자 : 0

[+ 증가] [- 감소]

버튼을 누르면 숫자가 자동으로 바뀐다.

즉, state가 바뀌면 화면이 새로 고쳐지지 않아도 React가 알아서 업데이트한다.

## ❖ 한눈에 정리

| 개념 | 설명 | 예시 |
|----|----|----|
|----|----|----|

|       |                   |                        |
|-------|-------------------|------------------------|
| JSX   | HTML처럼 화면을 만드는 문법 | <h1>{name}</h1>        |
| 컴포넌트  | 화면을 나누는 블록        | <Header />, <Footer /> |
| props | 부모 → 자식으로 값 전달    | <Welcome name="홍길동" /> |
| state | 변할 수 있는 값         | useState(0)            |

## 실습 문제

### ▣ 문제 1 : 자기소개 카드 만들기

새 Vite 프로젝트를 만들고 다음을 구현해보기

#### 요구사항

1. App.jsx → ProfileCard 컴포넌트 import
2. ProfileCard에 name, age, hobby props 전달
3. 화면에

안녕하세요, 저는 홍길동입니다.  
나이는 25살이고, 취미는 음악 감상입니다.

형태로 표시

#### 파일 구조

```
src/
  └── App.jsx
      └── ProfileCard.jsx
```

### ▣ 문제 2 : 좋아요 버튼 만들기

#### 요구사항

1. LikeButton 컴포넌트 생성
2. useState를 사용해서 클릭할 때마다 숫자가 1씩 증가하도록 구현
3. 초기값은 0, 버튼 이름은 “좋아요♥”

#### 결과 예시

좋아요 ♥ 3

(버튼을 클릭할 때마다 숫자가 올라감)

## ❶ 핵심 정리

React는 “데이터(state)”와 “화면(JSX)”이 자동으로 연결된다.

데이터를 바꾸면 화면이 새로 고침 없이 바뀐다.

이것이 HTML과 가장 큰 차이점이다

# Day 2 — useState 심화, 이벤트 핸들링, props 응용

## 오늘의 목표

- ✓ useState로 “바뀌는 데이터”를 다룰 수 있고
- ✓ 사용자의 행동(버튼 클릭, 입력 등)에 반응하는 방법을 이해하며
- ✓ props로 부모 → 자식 간 데이터 전달을 복습한다.

## 1. 이벤트(Event)란?

■ 이벤트 = “사용자의 행동”

버튼 클릭 → onClick

키보드 입력 → onChange

마우스 이동 → onMouseMove

리액트에서는 HTML과 다르게 이벤트 이름이 카멜표기법(소문자+대문자)으로 작성된다.

예 : onclick ✗ → onClick ☑

## 2. 버튼 클릭 이벤트 예제

```
// App.jsx
import { useState } from "react";

export default function App() {
  const [count, setCount] = useState(0);

  function increase() {
    setCount(count + 1);
  }

  return (
    <div>
      <h2>현재 숫자 : {count}</h2>
      <button onClick={increase}>+ 1 증가</button>
    </div>
  );
}
```

## 코드 설명

| 부분                  | 설명                    |
|---------------------|-----------------------|
| useState(0)         | count라는 state를 0으로 시작 |
| setCount(count + 1) | 버튼 누를 때마다 count를 1 증가 |
| onClick={increase}  | 클릭 시 increase 함수 실행   |

### 예상화면

현재 숫자 : 0  
[+ 1 증가]

버튼을 클릭할 때마다 숫자가 하나씩 올라간다.

## 3. 화살표 함수로 더 간단히 쓰기

리액트에서는 이벤트 함수를 짧게 작성할 수 있다.

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

○ 이렇게 쓰면 별도의 함수를 만들지 않아도 된다.  
(하지만 코드가 길어지면 가독성을 위해 따로 함수로 분리하는 게 좋다.)

## 4. 입력값 다루기 (onChange)

이번엔 사용자가 입력한 값을 화면에 바로 보여주는 예제

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

export default function App() {
  const [name, setName] = useState("");

  return (
    <div>
      <h2>이름을 입력하세요</h2>
      <input
        type="text"
        placeholder="이름 입력"
        onChange={(e) => setName(e.target.value)}
      />
      <p>안녕하세요, {name}님!</p>
    </div>
  );
}
```

## 💡 코드 설명

| 부분  | 설명                      |
|---|-------------------------|
| onChange={(e) => setName(e.target.value)} | 입력한 값이 바뀔 때마다 state에 저장 |
| {name}                                    | state의 내용을 바로 화면에 표시    |

### 💻 예상화면

이름을 입력하세요  
[홍길동]  
안녕하세요, 홍길동님!

- 한 글자 입력할 때마다 state가 갱신되어 화면도 실시간으로 바뀐다.

## 5. props 복습 : 부모 → 자식 데이터 전달

부모 컴포넌트(App)에서 자식 컴포넌트로 값을 전달

```
// Greeting.jsx
export default function Greeting(props) {
  return <h2>안녕하세요, {props.user}님</h2>;
}

// App.jsx
import Greeting from "./Greeting";

export default function App() {
  return (
    <div>
      <Greeting user="홍길동" />
      <Greeting user="이순신" />
    </div>
  );
}
```

### 💻 예상화면

안녕하세요, 홍길동님  
안녕하세요, 이순신님

- props는 “컴포넌트 간의 대화창구”라고 생각하자  
부모가 값을 주면, 자식이 그 값을 받아 화면에 출력한다.

## 6. 여러 개의 state 다루기

한 컴포넌트 안에서 여러 개의 데이터를 관리할 수 있다.

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

export default function App() {
  const [name, setName] = useState("");
  const [age, setAge] = useState("");

  return (
    <div>
      <h2>회원 정보 입력</h2>
      <input
        type="text"
        placeholder="이름 입력"
        onChange={(e) => setName(e.target.value)}
      />
      <input
        type="number"
        placeholder="나이 입력"
        onChange={(e) => setAge(e.target.value)}
      />
      <p>이름 : {name}</p>
      <p>나이 : {age}</p>
    </div>
  );
}
```

### 예상화면

회원 정보 입력

이름: [홍길동]

나이: [25]

이름 : 홍길동

나이 : 25

## 핵심 요약

| 개념       | 설명         | 예시                    |
|----------|------------|-----------------------|
| onClick  | 버튼 클릭 이벤트  | <button onClick={함수}> |
| onChange | 입력값 변경 이벤트 | <input onChange={함수}> |

|          |                |                                   |
|----------|----------------|-----------------------------------|
| useState | 화면 데이터 저장      | const [num, setNum] = useState(0) |
| props    | 부모 → 자식 데이터 전달 | <Child name="홍길동" />              |

○ 리액트의 원리

“사용자의 행동(Event) → state 변경 → 화면 자동 업데이트”

## 실습 문제 (새 프로젝트로 해보기)

### ▣ 문제 1 : 이름 저장 버튼 만들기

#### 요구사항

새 프로젝트 생성 후 App.jsx 파일 작성

input에 이름을 입력하고, 버튼을 누르면 이름이 화면에 표시되도록 만들기

입력창 아래에 "당신의 이름은 홍길동입니다" 형태로 표시

#### 힌트

useState("")

onChange로 입력값 관리

onClick으로 state 업데이트

#### 💡 예상화면

이름을 입력하세요

[홍길동]

[이름 저장]

당신의 이름은 홍길동입니다

### ▣ 문제 2 : 좋아요 버튼 만들기 (개선형)

#### 요구사항

useState로 숫자 상태 관리

버튼을 누를 때마다 좋아요 수 증가

10 이상이 되면 "많은 사랑을 받고 있습니다 ❤️ 문구 표시

#### 결과 예시

좋아요 수 : 9

[좋아요 ❤️

좋아요 수 : 10

많은 사랑을 받고 있습니다 ❤️

## ❸ 핵심 한 줄

“React는 사용자의 행동(Event)과 데이터(State)를 자동으로 연결한다.”

즉, 버튼을 누르거나 입력하면 → 데이터가 바뀌고 → 화면도 즉시 바뀐다.

# Day 3 – useEffect와 fetch (데이터 불러오기 기초)

## 오늘의 목표

- ✓ 리액트 컴포넌트가 언제 실행되고 다시 그려지는지(렌더링 시점) 이해한다.
- ✓ useEffect 흑(Hook)을 이용해 “화면이 처음 나타날 때” 코드를 실행한다.
- ✓ fetch() 함수를 사용해 외부에서 데이터를 불러와 화면에 표시한다.

## 1. 컴포넌트는 언제 실행될까?

리액트에서 컴포넌트는 “그려질 때(render)”마다 다시 실행된다.

즉, state가 바뀔 때마다 함수가 다시 실행되는 구조

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

export default function App() {
  const [count, setCount] = useState(0);
  console.log("컴포넌트 실행됨!");

  return (
    <div>
      <h2>카운터 : {count}</h2>
      <button onClick={() => setCount(count + 1)}>+1</button>
    </div>
  );
}
```

콘솔에는 버튼을 누를 때마다 “컴포넌트 실행됨!”이 반복 출력된다.

→ 리액트는 데이터(state)가 바뀔 때마다 화면 전체를 다시 그린다.

## 2. useEffect란?

리액트에서 “특정 시점”에 코드를 실행하고 싶을 때 쓰는 게 useEffect  
이건 “리액트의 생명주기(Lifecycle)”를 다루는 흑이다.

useEffect는 “컴포넌트가 처음 나타날 때”, 또는 “특정 데이터가 바뀔 때” 실행된다.

## 기본 문법

```
useEffect(() => {
  // 실행할 코드
}, []);
```

- () 안의 화살표 함수 : 실행할 동작
- [] 안의 의존성 배열(dependency array) : 언제 실행할지 결정

## 의존성 배열의 의미

| 형태        | 실행 시점              | 설명                   |
|-----------|--------------------|----------------------|
| []        | 화면이 처음 나타날 때 1번 실행 | “처음에만 실행”            |
| [state변수] | 지정한 변수가 바뀔 때마다 실행  | “특정 데이터가 바뀔 때 실행”    |
|           | 화면이 다시 그려질 때마다 실행  | “렌더링 될 때마다 실행” (주의!) |

## 3. useEffect 기초 예제

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

export default function App() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    console.log("처음 화면이 나타났어요!");
  }, []);

  return (
    <div>
      <h2>카운터 : {count}</h2>
      <button onClick={() => setCount(count + 1)}>+1</button>
    </div>
  );
}
```

콘솔에는 “처음 화면이 나타났어요!” 한 번만 출력된다.

→ []를 넣으면 컴포넌트가 처음 보일 때만 실행된다.

## 4. fetch()란?

 “외부에서 데이터를 가져오는 함수”

리액트는 백엔드(API 서버) 또는 공공데이터에서 정보를 가져와서 표시할 수 있다.

이때 사용하는 명령이 바로 `fetch()` 이다.

```
fetch("https://jsonplaceholder.typicode.com/users")
  .then((response) => response.json())
  .then((data) => console.log(data));
```

`fetch`는 데이터를 받아오면 `.then()` 안의 코드가 실행된다.

이게 바로 **비동기(asynchronous)** 처리이다.

(“잠시 기다렸다가 결과가 오면 실행하는 방식”)

## 5. useEffect + fetch 함께 쓰기

이제 `useEffect`를 이용해 화면이 처음 표시될 때 데이터를 가져오자.

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

export default function App() {
  const [users, setUsers] = useState([]);

  useEffect(() => {
    // 1. 화면이 처음 나타나면 fetch 실행
    fetch("https://jsonplaceholder.typicode.com/users")
      .then((response) => response.json()) // 2. 응답을 JSON으로 변환
      .then((data) => setUsers(data)); // 3. state에 저장
  }, []);

  return (
    <div>
      <h2>사용자 목록</h2>
      <ul>
        {users.map((user) => (
          <li key={user.id}>
            {user.name} ({user.email})
          </li>
        ))}
      </ul>
    </div>
  );
}
```

## 예상화면

사용자 목록

Leanne Graham (Sincere@april.biz)

Ervin Howell (Shanna@melissa.tv)

Clementine Bauch (Nathan@yesenia.net)

...

이 예시는 실제 무료 테스트용 API(jsonplaceholder)에서 사용자 데이터를 불러온다.  
(즉, 진짜 외부 서버에서 데이터를 가져온 거!)

## 6. 주의할 점

| 상황                | 설명   | 해결법                    |
|-------------------|--|------------------------|
| 무한<br>발생<br>fetch | useEffect 안에 []를 안 넣으면 화면이 다시 그려질 때마다 fetch 실행 | 항상 [] 넣기               |
| CORS 오류           | API 서버에서 React 접근을 허용하지 않은 경우                  | 다른 API 사용 or 서버 프록시 설정 |
| setState 오<br>타   | setUser → setUsers 실수 잊음                       | 변수 이름 정확히 확인           |

## 7. useEffect + state 응용 예제

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

export default function App() {
  const [time, setTime] = useState(new Date().toLocaleTimeString());

  useEffect(() => {
    const timer = setInterval(() => {
      setTime(new Date().toLocaleTimeString());
    }, 1000);
    return () => clearInterval(timer); // 컴포넌트 종료 시 타이머 제거
  }, []);

  return (
    <div>
      <h2>현재 시각 : {time}</h2>
    </div>
  );
}
```

setInterval로 1초마다 state가 갱신되므로 화면의 시간이 자동으로 변한다.  
이런 걸 “실시간 렌더링”이라고 한다.

## 핵심 정리

| 개념        | 설명                                 | 예시                                 |
|-----------|------------------------------------|------------------------------------|
| useEffect | 컴포넌트가 “처음 나타날 때” 또는 “데이터가 바뀔 때” 실행 | useEffect(() => {...}, [])         |
| fetch     | 외부 서버(API)에서 데이터를 불러오는 함수          | fetch("url").then(res=>res.json()) |
| 비동기       | 데이터가 도착하기 전에도 코드가 계속 실행되는 구조       | .then() 사용                         |

React는 “데이터가 들어오면 화면이 다시 그려진다”는 규칙을 기억!

## 실습 문제

### 문제 1 : 날씨 데이터 불러오기 (Open API 사용)

요구사항

fetch()로 아래 URL 데이터를 가져오기

<https://api.zippopotam.us/us/90210>

받아온 JSON 데이터 중 places[0].place name 값을 화면에 표시하기

제목 : “도시 이름 : Beverly Hills”

힌트

- useEffect로 fetch 실행
- useState로 데이터 저장
- places[0].["place name"] 접근

### 예상화면

도시 이름 : Beverly Hills

## ▣ 문제 2 : 게시글 목록 출력하기

요구사항

아래 주소의 데이터를 불러오기

<https://jsonplaceholder.typicode.com/posts>

제목(title)과 내용(body)을 &li&로 표시

상단에 “게시글 목록” 제목 출력

예시 화면

게시글 목록

1. sunt aut facere repellat provident occaecati...
  2. qui est esse...
  3. ea molestias quasi exercitationem repellat qui ipsa...
- ...

## ⓘ 한 줄 정리

**useEffect** : “언제 실행할지”를 정하는 도구

**fetch** : “어디서 데이터를 가져올지”를 정하는 도구

두 개를 합치면 “데이터가 들어오면 자동으로 화면이 만들어지는 리액트 앱” 완성 ☺





## 설치 단계 (React + Vite 기준)

### React + Vite 프로젝트 생성

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

### Tailwind CSS v4 및 공식 플러그인 설치

```
npm install -D tailwindcss @tailwindcss/vite
```

### vite.config.js 에 Tailwind 플러그인 추가

```
import { defineConfig } from 'vite'  
import react from '@vitejs/plugin-react'  
import tailwindcss from '@tailwindcss/vite'
```

```
export default defineConfig({  
  plugins: [  
    react(),  
    tailwindcss(),  
  ],  
})
```

### CSS 파일 준비 src/index.css

```
@import "tailwindcss";
```

```
npm run dev
```

## 색상 (Color)

| 분류   | 클래스 예시                                 | 설명                       |
|------|--|--------------------------|
| 배경색  | bg-blue-500, bg-gray-100, bg-green-700 | 배경색 지정 (색상명 + 단계 50~950) |
| 글자색  | text-white, text-black, text-red-600   | 글자 색상 지정                 |
| 테두리색 | border-gray-300, border-red-500        | 테두리(border) 색상           |
| 투명도  | bg-opacity-50, text-opacity-75         | 색상 투명도 지정                |
| 다크모드 | dark:bg-gray-800 dark:text-white       | 다크모드 활성 시 색상 전환          |

예시

```
<div className="bg-blue-500 text-white p-4 rounded">  
  파란색 배경, 흰색 글자  
</div>
```

## 글자 (Typography)

| 속성   | 클래스 예시   | 설명             |
|------|--|----------------|
| 크기   | text-xs ~ text-9xl   | 폰트 크기 단계별      |
| 두께   | font-thin, font-normal, font-semibold, font-bold, font-black | 글자 두께          |
| 스타일  | italic, not-italic   | 기울임 여부         |
| 정렬   | text-left, text-center, text-right, text-justify             | 텍스트 정렬         |
| 줄간격  | leading-none, leading-tight, leading-relaxed, leading-loose  | line-height    |
| 자간   | tracking-tighter, tracking-wide, tracking-wider              | letter-spacing |
| 대소문자 | uppercase, lowercase, capitalize                             | 글자 형태 변경       |
| 잘림처리 | truncate, text-ellipsis, overflow-clip                       | 글자 넘침 시 처리     |
| 폰트   | font-sans, font-serif, font-mono                             | 기본 폰트 패밀리 설정   |

예시

```
<h1 className="text-3xl font-bold text-center text-indigo-600 uppercase">
  Hello Tailwind
</h1>
```

## 여백 (Spacing)

| 구분          | 클래스 예시                         | 설명               |
|-------------|--------------------------------|------------------|
| 마진(Margin)  | m-4, mt-2, mb-8, mx-auto, my-4 | 바깥 여백            |
| 패딩(Padding) | p-4, py-2, px-6                | 안쪽 여백            |
| 음수 마진       | -m-2, -mt-4                    | 음수 여백 지정         |
| 간격(Gap)     | gap-2, gap-6, gap-x-4, gap-y-8 | Flex/Grid 아이템 간격 |

예시

```
<div className="m-4 p-4 bg-gray-200">박스</div>
```

## 크기 (Width / Height / Max / Min)

| 속성 | 클래스 예시                            | 설명         |
|----|-----------------------------------|------------|
| 너비 | w-10, w-1/2, w-full, w-screen     | width 설정   |
| 높이 | h-10, h-1/2, h-screen, h-auto     | height 설정  |
| 최소 | min-w-[200px], min-h-[100px]      | 최소 크기 설정   |
| 최대 | max-w-lg, max-h-96, max-w-[400px] | 최대 크기 설정   |
| 비율 | aspect-square, aspect-video       | 비율 유지 컨테이너 |

예시

```
<div className="w-1/2 h-40 bg-green-200"></div>
```

## 배치 (Layout / Flex / Grid)

| 속성     | 클래스 예시   | 설명             |
|--------|--|----------------|
| Flex   | flex, inline-flex  | flex 컨테이너 활성화  |
| 방향     | flex-row, flex-col, flex-wrap                                  | 주축 방향 설정       |
| 정렬(가로) | justify-start, justify-center, justify-between, justify-around | main-axis 정렬   |
| 정렬(세로) | items-start, items-center, items-end, items-stretch            | cross-axis 정렬  |
| 정렬(전체) | content-center, content-between                                | 여러 행 정렬        |
| Grid   | grid, grid-cols-2, grid-cols-3, grid-rows-4                    | grid layout 구성 |
| Gap    | gap-2, gap-x-4, gap-y-6  | 그리드 간격 설정      |
| Order  | order-1, order-2   | 요소 순서 변경       |

예시

```
<div className="flex justify-between items-center p-4 bg-gray-100">
  <div>왼쪽</div>
  <div>오른쪽</div>
</div>
```

## 배경 / 경계선 / 그림자

| 속성     | 클래스 예시  | 설명            |
|--------|---|---------------|
| 배경색    | bg-blue-500, bg-gradient-to-r from-indigo-500 to-purple-500 | 색상 및 그라데이션    |
| 테두리    | border, border-2, border-gray-300, border-dashed            | border 스타일    |
| 둥근 모서리 | rounded, rounded-lg, rounded-full                           | border-radius |
| 그림자    | shadow, shadow-md, shadow-xl, shadow-inner                  | box-shadow    |
| 불투명도   | opacity-50, opacity-100                                     | 투명도 조정        |
| 혼합 모드  | mix-blend-multiply, mix-blend-overlay                       | blend 모드 적용   |

예시

```
<button className="bg-gradient-to-r from-blue-500 to-indigo-600 text-white px-4 py-2 rounded-lg shadow-lg">
  로그인
</button>
```

## 테두리(Border)

| 속성 | 클래스 예시   | 설명        |
|----|--|-----------|
| 굵기 | border, border-2, border-4, border-t, border-b-4 | 테두리 두께    |
| 색상 | border-red-500, border-gray-200                  | 테두리 색상 지정 |

|         |   |                                 |
|---------|---|---------------------------------|
| 모양      | rounded-none, rounded-sm, rounded, rounded-md, rounded-lg, rounded-xl, rounded-2xl, rounded-3xl, rounded-full | 둥근 정도 조정                        |
| 스타일     | border-solid, border-dashed, border-dotted, border-double   | border 스타일 변경                   |
| 전체 테두리  | border, border-2, border-4, border-8  | 테두리 두께                          |
| 방향별     | border-t, border-b-2, border-l-4, border-r  | 방향별 테두리                         |
| 분리선     | divide-x, divide-y, divide-gray-200, divide-dashed  | flex/grid 자식 간 구분선              |
| outline | outline, outline-none, outline-2, outline-blue-400  | 포커스 외곽선                         |
| ring    | ring-2 ring-blue-400, ring-offset-2 ring-offset-gray-100  | 포커스용 링(Outline보다 부드럽고 그림자처럼 보임) |

## 목록 & 테이블

| 구분      | 클래스   | 설명           |
|---------|---|--------------|
| 리스트 스타일 | list-none, list-disc, list-decimal, list-inside, list-outside | 불릿/숫자 리스트 설정 |
| 테이블     | table-auto, table-fixed                                       | 테이블 레이아웃     |
| 행/열 구분선 | border-collapse, border-separate                              | 셀 경계 모드      |
| 셀 패딩    | px-4 py-2   | 테이블 셀 간격     |
| 줄무늬     | odd:bg-gray-50 even:bg-white                                  | 행별 색상 번갈아 적용 |

## 컬럼 (Multi-Column)

| 클래스   | 설명         |
|---|------------|
| columns-1~columns-12                                      | 열 개수 지정    |
| columns-auto  | 자동 열       |
| columns-[250px]   | 고정 열 너비    |
| break-before-auto, break-inside-avoid, break-after-column | 컬럼 내 나눔 제어 |

예시

```
<div class="columns-3 gap-4">
  <p>컬럼 1 내용</p>
  <p>컬럼 2 내용</p>
  <p>컬럼 3 내용</p>
</div>
```

## 그라데이션 (Gradient)

| 클래스 | 설명  |
|-----|---|
| 방향  | bg-gradient-to-t, bg-gradient-to-r, bg-gradient-to-bl |

|     |                |
|-----|----------------|
| 시작색 | from-blue-500  |
| 중간색 | via-purple-500 |
| 끝색  | to-pink-500    |

예시

```
<div className="bg-gradient-to-r from-indigo-500 via-purple-500 to-pink-500 h-32"></div>
```

## 투명도 / 블렌드 (Opacity / Blend)

| 클래스   | 설명       |
|---|----------|
| opacity-0 ~ opacity-100                                 | 투명도      |
| mix-blend-multiply, mix-blend-overlay, mix-blend-screen | 혼합 모드    |
| bg-blend-multiply                                       | 배경 혼합 모드 |

## 스크롤 & 오버플로 (Overflow / Scroll / Snap)

| 클래스     | 설명   |
|---------|--|
| 오버플로    | overflow-hidden, overflow-scroll, overflow-x-auto, overflow-y-auto |
| 스크롤바 숨김 | scrollbar-hide (플러그인 필요)   |
| 스크롤 스냅  | snap-x, snap-y, snap-start, snap-center, snap-mandatory            |
| 스크롤 위치  | scroll-smooth, scroll-auto   |

예시

```
<div className="overflow-y-scroll h-40 snap-y snap-mandatory">
  <div className="snap-start h-40 bg-blue-200">1</div>
  <div className="snap-start h-40 bg-green-200">2</div>
</div>
```

## 배경 확장 (Background Utilities)

| 클래스          | 설명  |
|--------------|---|
| 배경 위치        | bg-center, bg-top, bg-bottom, bg-left, bg-right |
| 배경 반복        | bg-no-repeat, bg-repeat-x, bg-repeat-y          |
| 배경 크기        | bg-cover, bg-contain, bg-auto                   |
| 배경 첨부        | bg-fixed, bg-local, bg-scroll                   |
| 배경 불투명도      | bg-opacity-50                                   |
| 배경 이미지 직접 지정 | bg-[url('/img/bg.png')]                         |

## 위치 / Z-index / Display 확장

| 클래스     | 설명  |
|---------|---|
| 위치      | absolute, relative, fixed, sticky           |
| 오프셋     | top-0, bottom-4, left-2, right-10, inset-0  |
| Z-index | z-0, z-10, z-50, z-[9999]                   |
| Display | block, inline-block, flex, grid, hidden     |
| 가시성     | visible, invisible                          |
| 객체 맞춤   | object-contain, object-cover, object-center |

## 필터 / 백드롭 (Filter / Backdrop)

| 클래스    | 설명   |
|--------|--|
| 블러     | blur-sm, blur-md, blur-lg, blur-2xl                          |
| 밝기     | brightness-50, brightness-125                                |
| 대비     | contrast-150   |
| 색조 회전  | hue-rotate-90  |
| 반전     | invert   |
| 그레이스케일 | grayscale, sepia   |
| 투명도    | opacity-50   |
| 배경 필터  | backdrop-blur, backdrop-contrast-125, backdrop-brightness-75 |

예시

```
<div className="backdrop-blur-md bg-white/30 rounded-xl p-6">
  유리 느낌 카드
</div>
```

## 애니메이션 (Animation)

| 클래스                        | 설명            |
|----------------------------|---------------|
| animate-spin               | 회전            |
| animate-ping               | 확산 효과         |
| animate-bounce             | 튀는 애니메이션      |
| animate-pulse              | 천천히 깜빡임       |
| motion-safe:animate-bounce | 사용자의 모션 설정 존중 |

## 커서 / 선택 / 포인터

| 클래스     | 설명  |
|---------|---|
| 커서      | cursor-pointer, cursor-not-allowed, cursor-wait, cursor-crosshair |
| 선택      | select-none, select-text, select-all, select-auto                 |
| 포인터 이벤트 | pointer-events-none, pointer-events-auto                          |

## 트랜지션 (Transition)

| 클래스  | 설명  |
|------|---|
| 속성   | transition, transition-colors, transition-transform   |
| 지속시간 | duration-75, duration-150, duration-300, duration-700 |
| 이징   | ease-linear, ease-in, ease-out, ease-in-out           |
| 지연   | delay-100, delay-200                                  |

## 추가: 컨테이너 쿼리 (Container Queries — v4 신규)

| 클래스                                    | 설명                     |
|--|------------------------|
| container                              | 컨테이너 선언                |
| @container                             | 내부 조건부 스타일링 (CSS에서 사용) |
| size, min-width, aspect-ratio 기반 조건 가능 |                        |

예시

```
@container (min-width: 500px) {
  .card { @apply bg-blue-100; }
}
```

## 효과 / 전환 / 트랜스폼

| 속성         | 클래스 예시  | 설명          |
|------------|---|-------------|
| Hover      | hover:bg-blue-600, hover:scale-105                        | 마우스 오버 효과   |
| Focus      | focus:ring-2 focus:ring-blue-400                          | 포커스 효과      |
| Transition | transition, duration-300, ease-in-out                     | 애니메이션 속도 제어 |
| Transform  | scale-110, rotate-45, translate-x-4, skew-y-2             | 변형 효과       |
| Animation  | animate-bounce, animate-spin, animate-pulse, animate-ping | 기본 제공 애니메이션 |

예시

```
<button className="bg-pink-500 hover:scale-110 transition duration-300 ease-in-out text-white px-4 py-2 rounded">
  마우스 올려보세요
</button>
```

## 반응형 (Responsive)

| 접두사  | 조건 (min-width) | 예시           |
|------|----------------|--------------|
| sm:  | 640px          | sm:text-sm   |
| md:  | 768px          | md:text-lg   |
| lg:  | 1024px         | lg:text-xl   |
| xl:  | 1280px         | xl:text-2xl  |
| 2xl: | 1536px         | 2xl:text-3xl |

예시

```
<div className="bg-gray-200 sm:bg-green-200 md:bg-yellow-200 lg:bg-blue-200">
  화면 크기에 따라 색이 변합니다.
</div>
```

## 다크 모드 (Dark Mode)

| 클래스 예시               | 설명              |
|----------------------|-----------------|
| dark:bg-gray-900     | 다크모드 시 배경색 변경   |
| dark:text-white      | 다크모드 시 글자색 변경   |
| dark:border-gray-700 | 다크모드용 border 색상 |

예시

```
<div className="bg-white text-black dark:bg-gray-900 dark:text-white p-4">
  라이트 / 다크 모드 모두 대응
</div>
```

## 위치 (Position)

| 속성      | 클래스 예시                                    | 설명          |
|---------|---|-------------|
| 포지션     | static, relative, absolute, fixed, sticky | position 속성 |
| 오프셋     | top-0, bottom-4, left-2, right-10         | 위치 지정       |
| Z-Index | z-0, z-10, z-50, z-[9999]                 | 요소 우선순위 설정  |

## 표시 / 가시성 (Display / Visibility)

| 클래스   | 설명            |
|---|---------------|
| block, inline-block, inline                     | display 형식    |
| hidden  | 요소 숨김         |
| visible, invisible                              | visibility 제어 |
| overflow-hidden, overflow-auto, overflow-scroll | 넘침 제어         |

## 필터 (Filter & Backdrop)

| 속성    | 클래스 예시                                | 설명     |
|-------|---------------------------------------|--------|
| Blur  | blur-sm, blur-md, blur-lg             | 블러 효과  |
| 밝기    | brightness-50, brightness-125         | 밝기 조절  |
| 투명도   | opacity-75                            | 투명도    |
| 대비    | contrast-150                          | 대비 조절  |
| 배경 필터 | backdrop-blur, backdrop-brightness-75 | 뒷배경 효과 |

예시

```
<div className="backdrop-blur-md bg-white/30 p-6 rounded-xl">
  유리 효과 박스
</div>
```

## 기타 자주 쓰는 클래스

| 클래스                  | 설명              |
|----------------------|-----------------|
| cursor-pointer       | 마우스 포인터         |
| select-none          | 텍스트 드래그 금지      |
| shadow-lg            | 그림자 추가          |
| ring-2 ring-blue-400 | focus ring 추가   |
| divide-y, divide-x   | 구분선 추가          |
| space-y-4, space-x-2 | 자식 간 간격         |
| transition-colors    | 색상 전환만 애니메이션 적용 |

## 실제 종합 예시

```
function Card() {
  return (
```

```
<div className="p-6 max-w-sm bg-white dark:bg-gray-800 rounded-xl shadow-lg transition-all hover:scale-105">
  
  <h2 className="mt-4 text-xl font-bold text-center text-gray-800 dark:text-white">박문수</h2>
  <p className="text-center text-gray-500 dark:text-gray-300 text-sm">
    DW 아카데미 React 교강사
  </p>
  <button className="mt-4 w-full bg-indigo-500 hover:bg-indigo-600 text-white py-2 rounded">
    팔로우
  </button>
</div>
);
}
```