처음 배우는 훈련생을 위한 react 가이드

by JKR

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
  - HTML문서내에 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 하지않아도 바로 사용가능
  - 사용자 이벤트 제어 가능
  - DOM 사용가능
  - 클라이언트 측 객체지향 스크립트 언어
- 자바스크립트의 활용

  자바스크립트의 삽입 : 인라인 형태, 스크립트 태그를 사용하는 형태, 외부 파일을 지정하는 방법

  ○ 인라인 형태 : 이벤트와 함께 사용

  -
  - test

  ○ 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 = weight / (height * height) 계산**
- **BMI 값 출력**

2. 장바구니 합계 계산

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

- **총합을 구해 출력**
- **만약 총합 >= 20000 이면 "무료배송" 출력**
- **아니면 "배송비 3000원 추가" 출력**

3. 짝수/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 될수 있음 → 변수에 저장 가능

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

- 함수 표현식은 호이스팅이 되지 않음 → 반드시 정의후 호출해야함 (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 반환시 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.hasOwn(user, "age")); // true/false (권장)

// 안전 접근(옵셔널 체이닝)
const city = user.address?.city;       // address가 없으면 undefined 반환, 에러 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 Array객체로, 순서가 있는 컬렉션을 표현한다.

■ 여러값을 하나의 변수로 묶어서 다룰 수 있게 해주는 자료 구조이다.

■ 컬렉션이란 여러개의 항목들을 모아놓은 것, 배열은 항목드의 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];              // 얕은 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
- 매열 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)