

**개발자를 위한**

**JavaScript, jQuery, AJAX, React.js**

*Y-A,Dominica KIM*

# 목차

1

**JavaScript 소개 및 기본 문법**

변수, 함수, 객체, 배열, 이벤트 처리

2

**jQuery 활용법**

선택자, 이벤트, 애니메이션, DOM 조작

3

**AJAX 이해와 실습**

비동기 통신, JSON, API 연동

4

**React.js 원리와 실전**

컴포넌트, JSX, 상태 관리, 라우팅

# JavaScript란?



## 웹의 표준 프로그래밍 언어

모든 현대 브라우저에서 지원되는 언어입니다. 웹사이트의 동작을 담당합니다.



## 동적 웹 페이지 구현의 핵심

사용자 상호작용과 실시간 업데이트를 가능하게 합니다.



## 다양한 프레임워크의 기반

jQuery, React, Angular, Vue 등의 기초가 됩니다.



## 역사와 발전

1995년 넷스케이프에서 브렌던 아이크가 개발했으며, ES6(2015), ES2016~ES2023 등 꾸준한 버전 업데이트로 발전해왔습니다.

# 자바스크립트 온라인 리소스

자바스크립트 학습에 도움이 되는 유용한 온라인 자료들입니다.

사이트명	유형	설명	URL
MDN Web Docs	공식 문서	Mozilla에서 제공하는 웹 기술 문서로, 자바스크립트 레퍼런스와 튜토리얼을 포함합니다.	developer.mozilla.org
W3Schools	튜토리얼	초보자에게 친숙한 자바스크립트 기초 학습 사이트입니다.	w3schools.com
JavaScript.info	튜토리얼/가이드	기초부터 고급까지 현대 자바스크립트를 배울 수 있는 상세한 가이드입니다.	javascript.info
freeCodeCamp	실습/프로젝트	무료 코딩 학습 플랫폼으로, 자바스크립트 실습과 프로젝트를 제공합니다.	freecodecamp.org
Codecademy	인터랙티브 코스	인터랙티브한 자바스크립트 과정을 통해 실시간으로 코딩을 배울 수 있습니다.	codecademy.com
Udemy	온라인 강의	다양한 나이도와 주제의 자바스크립트 강의를 제공하는 플랫폼입니다.	udemy.com

# 무료 가상 REST API 서비스

서비스명	설명	URL
JSONPlaceholder	간단한 가짜 데이터를 제공하는 대표적인 REST API 서비스입니다.	<a href="http://jsonplaceholder.typicode.com">jsonplaceholder.typicode.com</a>
Reqres	사용자 데이터와 리소스를 제공하는 API로 실제 응답을 시뮬레이션합니다.	<a href="http://reqres.in">reqres.in</a>
MockAPI	커스텀 API를 직접 생성하고 관리할 수 있는 서비스입니다.	<a href="http://mockapi.io">mockapi.io</a>
DummyJSON	제품, 카트, 사용자 등 다양한 더미 데이터를 제공합니다.	<a href="http://dummyjson.com">dummyjson.com</a>
Public APIs	다양한 카테고리의 무료 공개 API를 모아둔 딕레토리입니다.	<a href="https://github.com/public-apis/public-apis">github.com/public-apis/public-apis</a>
REST Countries	국가 정보를 제공하는 API로 실제 데이터를 활용할 수 있습니다.	<a href="http://restcountries.com">restcountries.com</a>
Postman	API 개발 및 테스트를 위한 협업 플랫폼으로, 내장된 모의 서버 기능을 제공합니다.	<a href="http://postman.com">postman.com</a>

# 변수 선언 (var, let, const)

## var

함수 스코프를 가집니다. 호이스팅이 발생합니다.

```
var name = "Dominica";
if (true) {
  var name = "RuRi";
}
console.log(name); // RuRi
```

## let

블록 스코프를 가집니다. 재할당 가능합니다.

```
let age = 25;
if (true) {
  let age = 30;
}
console.log(age); // 25
```

## const

블록 스코프를 가집니다. 재할당이 불가능합니다.

```
const PI = 3.14;
// PI = 3.15; // 에러 발생
console.log(PI); // 3.14
```

### 선언 방식

### 스코프 종류

### 예시 결과

var	함수 스코프	바깥 변수 덮음 (RuRi)
let/const	블록 스코프	바깥 변수 유지 (Dominica)

# 자바스크립트 데이터 타입

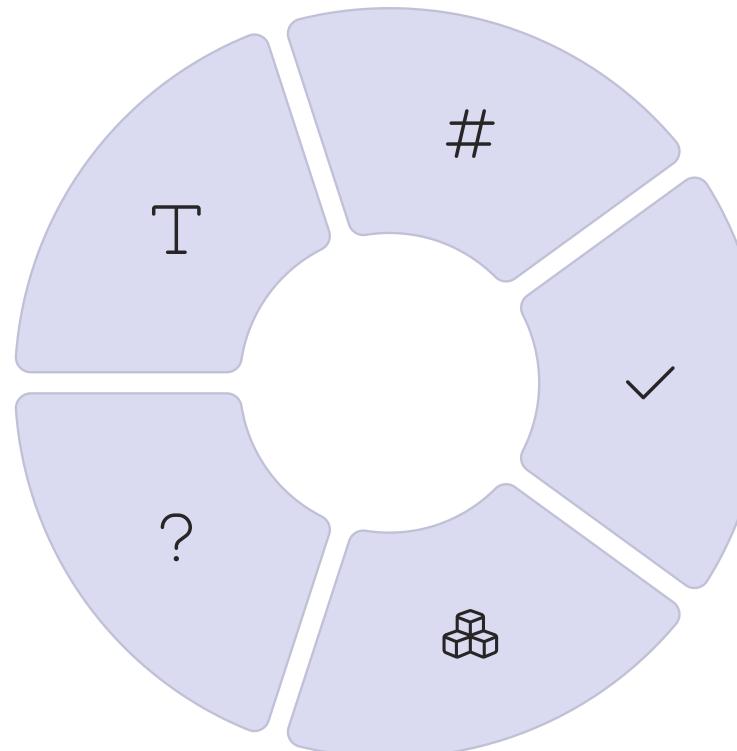
## 문자열 (String)

텍스트 데이터를 저장합니다.

'안녕하세요', "JavaScript"

## 특수 값

null, undefined



## 숫자 (Number)

정수와 실수를 모두 포함합니다.

42, 3.14, NaN, Infinity

## 불리언 (Boolean)

참/거짓 값을 나타냅니다.

true, false

## 객체 (Object)

복합 데이터 타입입니다.

{}, [], function(){}  
etc.

# 자바스크립트 연산자

산술 연산자	<code>+, -, *, /, %, **</code>	<code>5 + 3 = 8, 10 % 3 = 1</code>
할당 연산자	<code>=, +=, -=, *=, /=</code>	<code>x += 5</code> 는 <code>x = x + 5</code> 와 동일
비교 연산자	<code>==, ===, !=, !==, &gt;, &lt;, &gt;=, &lt;=</code>	<code>5 === "5"</code> 는 false (타입까지 비교)
논리 연산자	<code>&amp;&amp;,   , !</code>	<code>true &amp;&amp; false</code> 는 false
삼항 연산자	<code>조건 ? 참값 : 거짓값</code>	<code>age &gt;= 18 ? "성인" : "미성년자"</code>

# 자바스크립트 함수

함수 선언식

function 키워드로 정의, 호이스팅으로 선언 전 호출 가능

함수 표현식

변수에 함수 할당, 선언 후에만 사용 가능

화살표 함수

ES6의 간결한 구문, this 바인딩 없음

```
// 함수 선언식
```

```
function add(a, b) { return a + b; }
```

```
// 함수 표현식
```

```
const multiply = function(a, b) { return a * b; };
```

```
// 화살표 함수
```

```
const square = x => x * x;
```

# 자바스크립트 조건문

## if...else 문

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

## switch 문

```
switch (day) {  
    case 0:  
        console.log("일요일");  
        break;  
    case 1:  
        console.log("월요일");  
        break;  
    default:  
        console.log("다른 요일");  
}
```

## 삼항 연산자

```
const message = age >= 18  
    ? "투표할 수 있습니다"  
    : "투표할 수 없습니다";  
  
console.log(message);
```

# 자바스크립트 반복문

## for 루프

```
for (let i = 0; i < 5; i++) {  
    console.log(i); // 0, 1, 2, 3, 4  
}
```

## while 루프

```
let i = 0;  
while (i < 5) {  
    console.log(i); // 0, 1, 2, 3, 4  
    i++;  
}
```

## for...of 루프

```
const fruits = ["사과", "바나나", "딸기"];  
for (const fruit of fruits) {  
    console.log(fruit); // 사과, 바나나, 딸기  
}
```

## forEach 메서드

```
const numbers = [1, 2, 3, 4, 5];  
numbers.forEach(function(number) {  
    console.log(number * 2); // 2, 4, 6, 8, 10  
});
```

# 자바스크립트 배열

메서드	설명	예시
push()	배열의 끝에 요소를 추가합니다	<code>fruits.push("오렌지");</code>
pop()	배열의 마지막 요소를 제거하고 반환합니다	<code>fruits.pop();</code>
unshift()	배열의 앞에 요소를 추가합니다	<code>fruits.unshift("레몬");</code>
shift()	배열의 첫 번째 요소를 제거하고 반환합니다	<code>fruits.shift();</code>
map()	모든 요소에 함수를 적용하여 새 배열을 생성합니다	<code>const doubled = numbers.map(n =&gt; n * 2);</code>
reduce()	배열의 모든 요소를 하나의 값으로 줄입니다	<code>const sum = numbers.reduce((a, b) =&gt; a + b, 0);</code>
filter()	조건을 만족하는 요소만 새 배열로 반환합니다	<code>const evens = numbers.filter(n =&gt; n % 2 === 0);</code>

메서드	설명	예시
slice()	배열의 일부분을 추출하여 새 배열로 반환합니다	<code>const part = fruits.slice(1, 3);</code>
splice()	배열에서 요소를 추가/제거하고 제거된 요소를 반환합니다	<code>fruits.splice(1, 1, "키위");</code>
join()	배열의 모든 요소를 문자열로 결합합니다	<code>fruits.join(", ");</code>
concat()	두 개 이상의 배열을 결합합니다	<code>const newArray = arr1.concat(arr2);</code>
forEach()	배열의 모든 요소에 함수를 실행합니다	<code>fruits.forEach(fruit =&gt; console.log(fruit));</code>
find()	조건을 만족하는 첫 번째 요소를 반환합니다	<code>const found = users.find(user =&gt; user.id === 1);</code>
includes()	배열에 특정 요소가 포함되어 있는지 확인합니다	<code>const hasApple = fruits.includes("사과");</code>

# 자바스크립트 객체

## 객체 리터럴

```
const person = {  
  name: "김철수",  
  age: 30,  
  job: "개발자",  
  sayHello: function() {  
    return `안녕하세요, ${this.name}입니다.`;  
  }  
};
```

## 객체 접근

```
// 점 표기법  
console.log(person.name); // 김철수  
  
// 대괄호 표기법  
console.log(person["age"]); // 30  
  
// 메서드 호출  
console.log(person.sayHello()); // 안녕하세요, 김철수입니다.
```

객체는 키-값 쌍의 컬렉션입니다. 속성과 메서드를 포함할 수 있습니다.

# 자바스크립트 이벤트

이벤트 리스너 등록

요소에 이벤트 핸들러를 추가합니다.

이벤트 발생

사용자 동작이 이벤트를 트리거합니다.

이벤트 처리

등록된 핸들러가 실행됩니다.

```
// 이벤트 리스너 등록
const button = document.querySelector("#myButton");
button.addEventListener("click", function(event) {
  console.log("버튼이 클릭되었습니다!");
  console.log(event.target);
});
```

# 자바스크립트 이벤트 종류

마우스 이벤트	click, dblclick, mousedown, mouseup, mousemove, mouseover, mouseout, mouseenter, mouseleave
키보드 이벤트	keydown, keyup, keypress
문서/윈도우 이벤트	load, unload, resize, scroll, DOMContentLoaded
폼 이벤트	submit, reset, change, focus, blur, input
터치 이벤트	touchstart, touchmove, touchend, touchcancel

# 자바스크립트 주요 내장클래스

자바스크립트에서 자주 사용되는 유용한 내장클래스들입니다.

클래스 종류	클래스명	설명	예시
문자열 클래스	String	문자열을 다루기 위한 객체	<code>new String("Hello")</code>
배열 클래스	Array	배열을 생성하고 조작하는 객체	<code>new Array(1,2,3)</code>
객체 클래스	Object	모든 객체의 기본 클래스	<code>new Object() 또는 {}</code>
날짜 클래스	Date	날짜와 시간을 다루는 객체	<code>new Date()</code>
수학 클래스	Math	수학적 연산을 위한 정적 객체	<code>Math.PI, Math.random()</code>
정규식 클래스	RegExp	정규 표현식을 위한 객체	<code>new RegExp('\\w+')</code>
맵 클래스	Map	키-값 쌍을 저장하는 컬렉션	<code>new Map([('key', 'value')])</code>
집합 클래스	Set	고유한 값을 저장하는 컬렉션	<code>new Set([1, 2, 3])</code>
프로미스 클래스	Promise	비동기 작업 결과를 처리하는 객체	<code>new Promise((resolve)=&gt;{})</code>
오류 클래스	Error	런타임 오류를 나타내는 객체	<code>new Error('오류 메시지')</code>

이러한 내장클래스들은 자바스크립트 프로그래밍에서 다양한 데이터 타입과 기능을 구현하는데 필수적입니다.

# 자바스크립트 DOM 조작

## DOM 요소 선택

`getElementById()`

ID로 단일 요소 선택

`getElementsByClassName()`

클래스명으로 여러 요소 선택

`getElementsByTagName()`

태그명으로 여러 요소 선택

`querySelector()`

CSS 선택자로 단일 요소 선택

`querySelectorAll()`

CSS 선택자로 여러 요소 선택

## DOM 요소 조작

`innerHTML`

요소 내부 HTML 변경

`textContent`

요소 내부 텍스트 변경

`setAttribute()`

속성 설정

`getAttribute()`

속성 값 가져오기

`style.property`

CSS 스타일 변경

`classList`

클래스 추가/제거/토글

## DOM 요소 생성 및 추가

`createElement()`

새 요소 생성

`createTextNode()`

텍스트 노드 생성

`appendChild()`

자식 요소로 추가

`insertBefore()`

특정 위치에 삽입

`removeChild()`

자식 요소 제거

`replaceChild()`

자식 요소 교체

# 자바스크립트 DOM 조작

## 요소 선택

```
// ID로 선택  
const title =  
document.getElementById("title");  
  
// CSS 선택자로 선택  
const paragraph =  
document.querySelector(".content");  
const items =  
document.querySelectorAll("li");
```

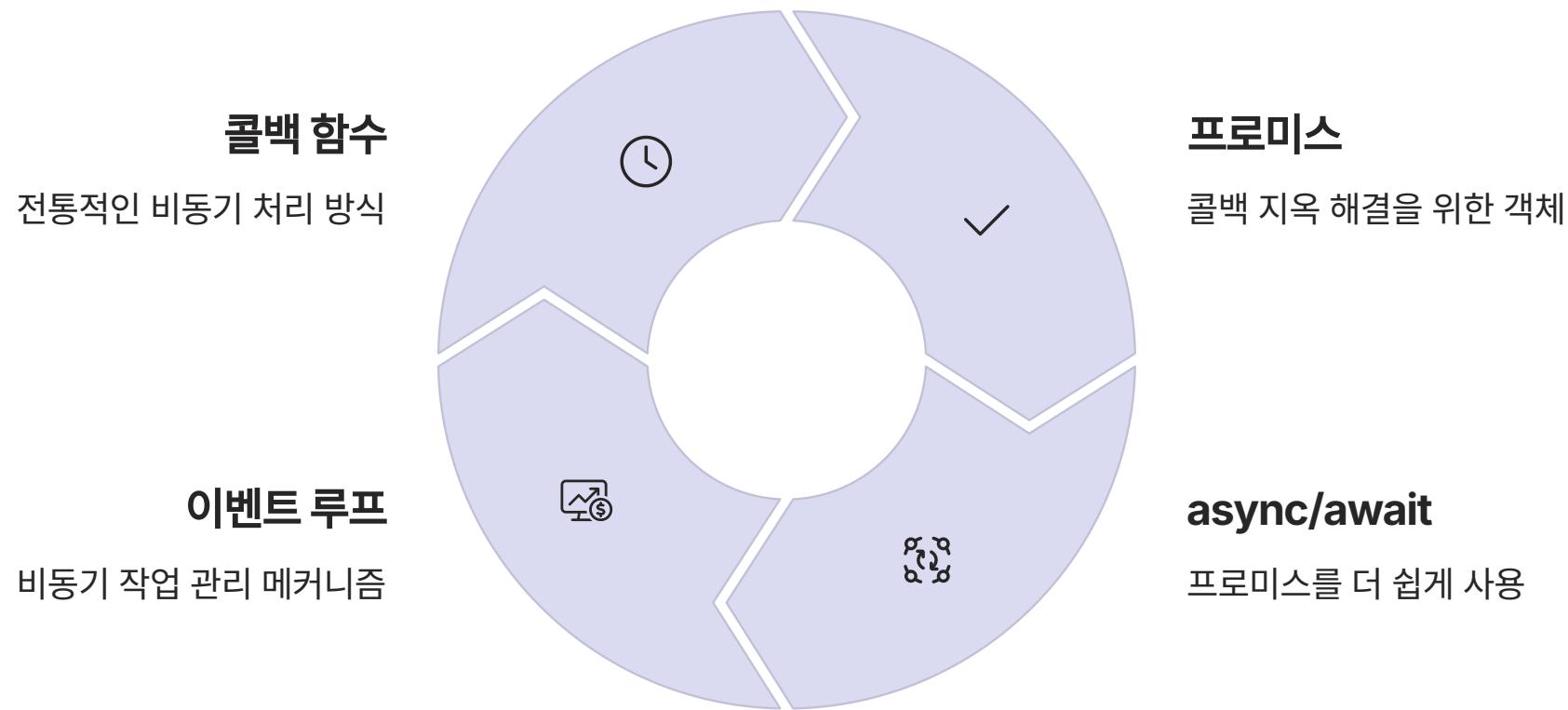
## 내용 변경

```
// 텍스트 내용 변경  
title.textContent = "새로운 제목";  
  
// HTML 내용 변경  
paragraph.innerHTML = "새로운 내용";
```

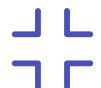
## 스타일 및 속성 변경

```
// 스타일 변경  
title.style.color = "blue";  
title.style.fontSize = "24px";  
  
// 속성 변경  
const link =  
document.querySelector("a");  
link.setAttribute("href",  
"https://example.com");
```

# 자바스크립트 비동기 프로그래밍



# jQuery 소개



## 간결한 문법

복잡한 JavaScript 작업을 간단하게 처리합니다.



## 크로스 브라우저 지원

브라우저 호환성 문제를 해결합니다.



## 풍부한 플러그인

확장성이 뛰어나며 다양한 기능을 추가할 수 있습니다.



## 애니메이션 효과

시각적 효과를 쉽게 구현할 수 있습니다.

# jQuery 설치 및 기본 구문

## jQuery 추가하기

```
<!-- CDN으로 jQuery 추가 -->
<script src="https://code.jquery.com/jquery-3.7.1.min.js">
</script>

<!-- 또는 로컬 파일로 추가 -->
<script src="js/jquery.min.js"></script>
```

## 기본 구문

```
// jQuery 기본 구문
$(선택자).동작();

// 예제
$("p").hide(); // 모든 p 요소를 숨김
$("#id").show(); // id가 'id'인 요소를 표시
```

## 문서 준비

```
// 문서가 준비되면 실행
$(document).ready(function() {
    // jQuery 코드
});

// 짧은 문법
$(function() {
    // jQuery 코드
});
```

# jQuery 선택자

기본 선택자	<code>\$("#id"), \$(".class"), \$("element")</code>	ID, 클래스, 태그 이름으로 선택
계층 선택자	<code>("parent &gt; child"), ("ancestor descendant")</code>	부모-자식 관계로 선택
필터 선택자	<code>("li:first"), ("input:checked"), ("p:odd")</code>	특정 조건에 맞는 요소 선택
속성 선택자	<code>("a[href]"), ("input[type='text']")</code>	특정 속성이 있는 요소 선택
복합 선택자	<code>("selector1, selector2, selector3")</code>	여러 선택자를 쉼표로 결합

# jQuery DOM 탐색

## 상위 요소 탐색

- `parent()` - 직계 부모
- `parents()` - 모든 조상
- `closest()` - 조건에 맞는 가장 가까운 조상

## 하위 요소 탐색

- `children()` - 직계 자식
- `find()` - 모든 후손

## 형제 요소 탐색

- `siblings()` - 모든 형제
- `next()` - 다음 형제
- `prev()` - 이전 형제

```
// DOM 탐색 예제
```

```
$( "li.active" ).parent(); // active 클래스를 가진 li의 부모  
$( "ul" ).children( "li" ); // ul의 직계 자식 중 li 요소들  
$( "h1" ).siblings( "p" ); // h1의 형제 중 p 요소들
```

# jQuery DOM 조작

jQuery를 사용하여 DOM 요소의 내용, 위치, 속성을 조작하는 다양한 방법

카테고리	메소드	설명 및 예제
내용 조작	text()	텍스트 내용 가져오기: <code>\$(“p”).text();</code>
	text(value)	텍스트 설정: <code>\$(“p”).text(“새로운 텍스트”);</code>
	html()	HTML 내용 가져오기: <code>\$(“div”).html();</code>
	html(value)	HTML 설정: <code>\$(“div”).html(“새 HTML”);</code>

# jQuery DOM 조작 위치

카테고리	메소드	설명
요소 위치 정보 가져오기	offset()	문서 기준 위치 좌표 반환
요소 위치 정보 가져오기	position()	부모 요소 기준 위치 좌표 반환
DOM 트리 위치 조작	appendTo()	선택한 요소 내부 끝에 추가
DOM 트리 위치 조작	prependTo()	선택한 요소 내부 앞에 추가
DOM 트리 위치 조작	insertBefore()	선택한 요소 앞에 삽입
DOM 트리 위치 조작	insertAfter()	선택한 요소 뒤에 삽입
실전 사용 예제	<code> \$("p").offset();</code>	문서 내 절대 위치 좌표 반환
실전 사용 예제	<code>\$("div").append("#container");</code>	새 div 요소를 #container 끝에 추가
실전 사용 예제	<code> \$("p").prepend("처음");</code>	새 span 요소를 p 요소 앞에 추가

# jQuery DOM 속성

jQuery를 사용하여 DOM 요소의 속성(attribute)과 프로퍼티(property)를 조작하는 다양한 방법

메소드	설명	예제
attr(name)	HTML 속성 값 가져오기	<code>\$( "img" ).attr("src");</code>
attr(name, value)	HTML 속성 값 설정하기	<code>\$( "img" ).attr("src", "new.jpg");</code>
removeAttr(name)	HTML 속성 제거하기	<code>\$( "a" ).removeAttr("target");</code>
prop(name)	DOM 프로퍼티 값 가져오기	<code>\$( "input" ).prop("checked");</code>
prop(name, value)	DOM 프로퍼티 값 설정하기	<code>\$( "input" ).prop("disabled", true);</code>
addClass(class)	클래스 추가하기	<code>\$( "p" ).addClass("highlight");</code>
removeClass(class)	클래스 제거하기	<code>\$( "p" ).removeClass("highlight");</code>
toggleClass(class)	클래스 토글하기	<code>\$( "p" ).toggleClass("active");</code>

## attr()와 prop()의 차이점

- attr(): HTML 속성을 조작 (문서에 직접 표시되는 속성)
- prop(): DOM 프로퍼티를 조작 (JavaScript 객체 프로퍼티)

```
// 속성과 프로퍼티 예제
$( "a" ).attr("href", "https://jquery.com"); // href 속성 변경
$( "input" ).prop("checked", true); // checked 상태 변경
$( "div" ).addClass("highlight"); // 클래스 추가
```

# jQuery 이벤트 처리

## 이벤트 바인딩

```
// 클릭 이벤트
$("#button").click(function() {
  alert("버튼이 클릭되었습니다!");
});

// 마우스 이벤트
$p").mouseenter(function() {
  $(this).css("color", "red");
});
```

## on() 메서드

```
// 여러 이벤트 처리
$("#element").on({
  click: function() { console.log("클릭됨"); },
  mouseenter: function() { console.log("마우스 진입"); },
  mouseleave: function() { console.log("마우스 이탈"); }
});
```

## 이벤트 위임

```
// 동적으로 추가된 요소에도 이벤트 적용
$("#parent").on("click", "button", function() {
  console.log("동적으로 추가된 버튼도 작동합니다!");
});
```

## 이벤트 제거

```
// 특정 이벤트 제거
$("#button").off("click");

// 모든 이벤트 제거
$("#button").off();
```

# jQuery 효과와 애니메이션



## 표시/숨김 효과

```
// 기본 효과  
$("#element").hide(); // 숨기기  
$("#element").show(); // 표시하기  
$("#element").toggle(); // 토플(숨김/표시)
```



## 페이드 효과

```
// 페이드 효과  
$("#element").fadeIn(1000); // 1초 동안 페이드 인  
$("#element").fadeOut(500); // 0.5초 동안 페이드 아웃  
$("#element").fadeToggle(); // 페이드 토플
```



## 슬라이드 효과

```
// 슬라이드 효과  
$("#element").slideDown(); // 아래로 슬라이드  
$("#element").slideUp(); // 위로 슬라이드  
$("#element").slideToggle(); // 슬라이드 토플
```



## 사용자 정의 애니메이션

```
// 사용자 정의 애니메이션  
$("#element").animate({  
    opacity: 0.5,  
    left: "+=50",  
    height: "toggle"  
}, 1000, function() {  
    console.log("애니메이션 완료!");  
});
```

# jQuery와 CSS

## CSS 속성 조작

```
// CSS 속성 가져오기  
const color = $("#element").css("color");  
  
// 단일 속성 설정  
$("#element").css("color", "blue");  
  
// 여러 속성 설정  
$("#element").css({  
  "color": "white",  
  "background-color": "black",  
  "font-size": "16px"  
});
```

## 클래스 조작

```
// 클래스 추가  
$("#element").addClass("highlight");  
  
// 클래스 제거  
$("#element").removeClass("active");  
  
// 클래스 토클  
$("#element").toggleClass("selected");  
  
// 클래스 존재 확인  
if ($("#element").hasClass("error")) {  
  console.log("오류 상태입니다");  
}
```

# jQuery 실전 예제: 탭 인터페이스

단계	구성요소	설명
1	HTML 구조	탭과 콘텐츠 패널 마크업
2	CSS 스타일링	탭 디자인과 시각적 상태
3	jQuery 기능	이벤트 처리와 콘텐츠 전환

```
// HTML
<ul class="tabs">
  <li><a href="#tab1">탭 1</a></li>
  <li><a href="#tab2">탭 2</a></li>
</ul>
<div class="tab-content">
  <div id="tab1">첫 번째 탭 내용</div>
  <div id="tab2">두 번째 탭 내용</div>
</div>

// jQuery
$(".tabs a").on("click", function(e) {
  e.preventDefault();
  $(".tab-content > div").hide();
  $($this).attr("href")).show();
  $(".tabs a").removeClass("active");
  $(this).addClass("active");
});
```

# AJAX 소개



## AJAX란?

Asynchronous JavaScript and XML의 약자입니다. 페이지 새로고침 없이 서버와 통신하는 기술입니다.



## AJAX의 장점

페이지 리로드 없이 데이터를 가져와 사용자 경험을 향상시킵니다. 서버 부하도 줄여줍니다.



## 구현 방법

XMLHttpRequest 객체, jQuery의 \$.ajax(), fetch API 등으로 구현할 수 있습니다.

# 일반 JavaScript로 AJAX 구현

## AJAX 요청 설정 단계

### 1. XMLHttpRequest 객체 생성

```
// XMLHttpRequest 객체 생성  
const xhr = new XMLHttpRequest();
```

### 2. 요청 초기화 및 설정

```
// GET 요청 설정  
xhr.open("GET",  
        "https://api.example.com/data",  
        true);  
  
// 요청 헤더 설정 (필요한 경우)  
xhr.setRequestHeader("Content-  
Type", "application/json");
```

## AJAX 요청 실행 단계

### 3. 이벤트 핸들러 등록

```
// 응답 처리  
xhr.onload = function() {  
    if (xhr.status === 200) {  
        const response =  
            JSON.parse(xhr.responseText);  
        console.log(response);  
    } else {  
        console.error("오류 발생:",  
                    xhr.statusText);  
    }  
};
```

```
// 오류 처리  
xhr.onerror = function() {  
    console.error("요청 실패");  
};
```

### 4. 요청 전송

```
// 요청 보내기  
xhr.send();
```

# jQuery로 AJAX 구현

## \$.ajax() 메서드

```
// 기본 AJAX 요청
$.ajax({
  url: "https://api.example.com/data",
  method: "GET",
  dataType: "json",
  success: function(data) {
    console.log("데이터 수신 성공:", data);
  },
  error: function(xhr, status, error) {
    console.error("오류 발생:", error);
  }
});
```

## 단축 메서드

```
// GET 요청
$.get("https://api.example.com/data",
  function(data) {
    console.log(data);
  });

// POST 요청
$.post("https://api.example.com/create", {
  name: "홍길동",
  age: 30
}, function(response) {
  console.log("생성 완료:", response);
});
```

## Promise 방식

```
// jQuery 3.0 이상의 Promise 방식
$.ajax({
  url: "https://api.example.com/data"
})
.done(function(data) {
  console.log("성공:", data);
})
.fail(function(xhr, status, error) {
  console.error("실패:", error);
})
.always(function() {
  console.log("요청 완료");
});
```

# Fetch API를 이용한 AJAX

## 기본 Fetch 요청

```
// 기본 GET 요청
fetch("https://api.example.com/data")
  .then(response => response.ok ?
    response.json() :
    Promise.reject("네트워크 오류"))
  .then(data => console.log("데이터:", data))
  .catch(error => console.error("오류:", error));
```

## POST 요청 보내기

```
// POST 요청
fetch("https://api.example.com/create", {
  method: "POST",
  headers: {"Content-Type": "application/json"},
  body: JSON.stringify({
    name: "김철수",
    age: 25
  })
})
.then(response => response.json())
.then(data => console.log("생성:", data))
.catch(error => console.error("오류:", error));
```

## async/await 활용

```
// async/await 구문
async function fetchData() {
  try {
    const response = await fetch("https://api.example.com/data");
    if (!response.ok) throw new Error("네트워크 오류");
    const data = await response.json();
    console.log("데이터:", data);
  } catch (error) {
    console.error("오류:", error);
  }
}

fetchData();
```

# AJAX 데이터 형식: JSON

## JSON이란?

JavaScript Object Notation의 약자로, 데이터 교환을 위한 경량 형식입니다. 사람이 읽고 쓰기 쉽고 기계가 파싱하기 쉽습니다.

## JSON 문법

```
// JSON 예제
{
  "name": "홍길동",
  "age": 30,
  "isActive": true,
  "skills": ["JavaScript", "HTML", "CSS"],
  "address": {
    "city": "서울",
    "postcode": "12345"
  }
}
```

## JSON 변환

```
// JavaScript 객체를 JSON 문자열로 변환
const person = { name: "김철수", age: 25 };
const jsonString = JSON.stringify(person);
console.log(jsonString);
// 출력: {"name":"김철수","age":25}
```

```
// JSON 문자열을 JavaScript 객체로 변환
const obj = JSON.parse(jsonString);
console.log(obj.name); // 출력: 김철수
```

# AJAX 실전 예제: 데이터 로딩

## HTML 구조

```
<button id="loadData">데이터 불러오기</button>
<div id="results"></div>
```

## jQuery로 구현

```
$("#loadData").on("click", function() {
    // 로딩 표시
    $("#results").html("<p>데이터를 불러오는 중...</p>");

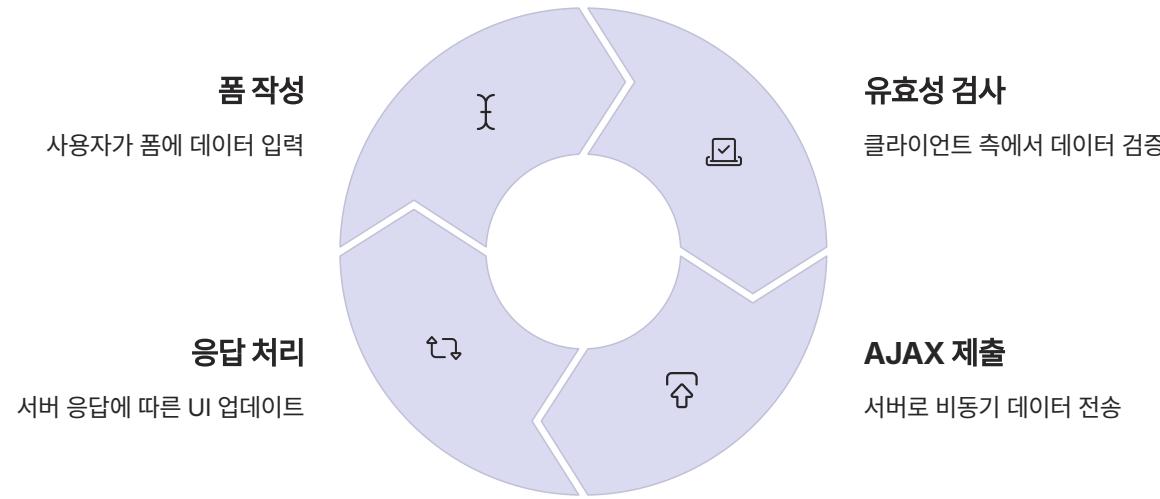
    // AJAX 요청
    $.ajax({
        url: "https://jsonplaceholder.typicode.com/users",
        method: "GET",
        dataType: "json",
        success: function(data) {
            let html = "<ul>";
            $.each(data, function(i, user) {
                html += `<li>${user.name} (${user.email})</li>`;
            });
            html += "</ul>";
            $("#results").html(html);
        },
        error: function() {
            $("#results").html("<p>데이터를 불러오는데 실패했습니다.</p>");
        }
    });
});
```

## Fetch API로 구현

```
document.getElementById("loadData").addEventListener("click", function() {
    const results = document.getElementById("results");
    results.innerHTML = "<p>데이터를 불러오는 중...</p>";

    fetch("https://jsonplaceholder.typicode.com/users")
        .then(response => response.json())
        .then(data => {
            let html = "<ul>";
            data.forEach(user => {
                html += `<li>${user.name} (${user.email})</li>`;
            });
            html += "</ul>";
            results.innerHTML = html;
        })
        .catch(error => {
            results.innerHTML = "<p>데이터를 불러오는데 실패했습니다.</p>";
        });
});
```

# AJAX 실전 예제: 폼 제출



```
// jQuery 폼 제출 예제
$("#contactForm").on("submit", function(e) {
  e.preventDefault(); // 기본 제출 동작 방지

  // 폼 데이터 수집
  const formData = {
    name: $("#name").val(),
    email: $("#email").val(),
    message: $("#message").val()
  };

  // AJAX 요청
  $.ajax({
    url: "https://api.example.com/contact",
    method: "POST",
    data: formData,
    success: function(response) {
      $("#formStatus").html("<p class='success'>메시지가 전송되었습니다!</p>");
      $("#contactForm")[0].reset(); // 폼 초기화
    },
    error: function() {
      $("#formStatus").html("<p class='error'>전송 중 오류가 발생했습니다.</p>");
    }
  });
});
```

# React.js 소개



## React란?

Facebook에서 개발한 UI 라이브러리입니다. 사용자 인터페이스를 구축하기 위한 JavaScript 라이브러리입니다.



## 컴포넌트 기반

재사용 가능한 컴포넌트로 UI를 구성합니다. 각 컴포넌트는 자체 상태와 렌더링 로직을 가집니다.



## 가상 DOM

실제 DOM 조작을 최소화하여 성능을 최적화합니다. 변경된 부분만 업데이트합니다.



## 선언적 문법

원하는 UI 상태를 선언하면 React가 DOM을 업데이트합니다. 복잡한 DOM 조작 코드가 불필요합니다.

# React 시작하기

## 1 Node.js 설치

React 개발을 위해 Node.js와 npm이 필요합니다.

```
// Node.js 버전 확인  
node -v  
npm -v
```

## 2 Create React App

새 React 프로젝트를 빠르게 시작하는 도구입니다.

```
// 새 프로젝트 생성  
npx create-react-app my-app  
  
// 프로젝트 폴더로 이동  
cd my-app
```

## 3 프로젝트 구조

주요 파일과 폴더構성을 이해합니다.

```
my-app/  
  README.md  
  node_modules/  
  package.json  
  public/  
    index.html  
    favicon.ico  
  src/  
    App.js  
    App.css  
    index.js  
    index.css
```

## 4 개발 서버 시작

```
// 개발 서버 시작  
npm start
```

# JSX 소개

## JSX란?

JavaScript XML의 약자로, JavaScript 내에서 HTML과 유사한 구문을 작성할 수 있게 해주는 React의 문법 확장입니다.

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

## JSX 규칙

- 모든 태그는 닫혀야 합니다
- 하나의 부모 요소로 감싸야 합니다
- JavaScript 표현식은 중괄호 {...}로 감쌉니다
- HTML 속성은 camelCase로 작성합니다

## JSX 활용 예

```
// JavaScript 표현식 사용
const name = "홍길동";
const element = <h1>안녕하세요, {name}님!</h1>;
```

```
// 속성 지정
const element = <img src={user.avatarUrl} alt="프로필" />;
```

```
// 조건부 렌더링
const element = (
  <div>
    {isLoggedIn ? (
      <h1>환영합니다!</h1>
    ) : (
      <button>로그인</button>
    )}
  </div>
);
```

# React 컴포넌트

## 컴포넌트 개념

재사용 가능한 독립적인 UI 조각

## 컴포넌트 유형

함수형 컴포넌트와 클래스 컴포넌트

## 컴포넌트 구성

계층적 컴포넌트 트리 형성

// 함수형 컴포넌트

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

// 클래스 컴포넌트

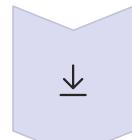
```
class Welcome extends React.Component {  
  render() {  
    return 안녕하세요, {this.props.name}님!;  
  }  
}
```

// 컴포넌트 사용

```
function App() {  
  return (
```

```
); }
```

# React Props



## Props란?

부모 컴포넌트에서 자식 컴포넌트로 데이터를 전달하는 방법입니다.



## Props 특징

읽기 전용이며 컴포넌트 내에서 수정할 수 없습니다.



## Props 활용

다양한 타입의 데이터와 함수를 전달할 수 있습니다.

```
// 부모 컴포넌트
function ParentComponent() {
  const handleClick = () => {
    alert('버튼이 클릭되었습니다!');
  };

  return (
    <div>
      <ChildComponent
        name="홍길동"
        age={30}
        isAdmin={true}
        hobbies={['독서', '영화 감상']}
        onClick={handleClick}
      />
    </div>
  );
}

// 자식 컴포넌트
function ChildComponent(props) {
  return (
    <div>
      <h2>{props.name}님, 안녕하세요!</h2>
      <p>나이: {props.age}세</p>
      <p>관리자 여부: {props.isAdmin ? '예' : '아니오'}</p>
      <p>취미: {props.hobbies.join(', ')})</p>
      <button onClick={props.onClick}>클릭</button>
    </div>
  );
}
```

# React State

컴포넌트의 상태 관리 방법

## 클래스 컴포넌트 (이전 버전 스타일)

this.state와 this.setState() 사용

```
// 클래스 컴포넌트의 state 예제
class TodoApp extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      items: [],
      text: ""
    };
  }

  handleChange = (e) => {
    this.setState({ text: e.target.value });
  }

  handleSubmit = (e) => {
    e.preventDefault();
    if (this.state.text.length === 0) return;

    const newItem = {
      text: this.state.text,
      id: Date.now()
    };

    this.setState(state => ({
      items: state.items.concat(newItem),
      text: ""
    }));
  }

  render() {
    return (

```

## 함수형 컴포넌트 (현재 React 주력 스타일)

useState() 등 Hooks 기반 사용

```
// 함수형 컴포넌트의 useState 흐 예제
function TodoApp() {
  const [items, setItems] = useState([]);
  const [text, setText] = useState("");

  const handleChange = (e) => {
    setText(e.target.value);
  };

  const handleSubmit = (e) => {
    e.preventDefault();
    if (text.length === 0) return;

    const newItem = {
      text: text,
      id: Date.now()
    };

    setItems([...items, newItem]);
    setText("");
  };

  return (

```

## 할 일 목록

{items.map(item => ( ))}

- {item.text}

추가

); }

## 할 일 목록

{this.state.items.map(item => ( ))}

- {item.text}

추가

); } }

개념	특징	클래스 컴포넌트	함수형 컴포넌트
State 정의	컴포넌트 내부에서 관리되는 데이터	this.state	useState()
State 변경	상태 업데이트 시 재렌더링	this.setState()	setter 함수 (ex: setItems)
생명주기	컴포넌트 생명주기와 연동	생명주기 메서드	useEffect() 흐

# React 이벤트 처리

## 이벤트 핸들링 특징

- camelCase 이벤트 이름 사용
- 함수를 이벤트 핸들러로 전달
- 기본 동작 방지를 위해 preventDefault() 사용

## 이벤트 바인딩

```
// 클래스 컴포넌트에서 이벤트 바인딩
class Toggle extends React.Component {
  constructor(props) {
    super(props);
    this.state = { isOn: false };
    // 바인딩 필요
    this.handleClick = this.handleClick.bind(this);
  }

  handleClick() {
    this.setState(prevState => ({
      isOn: !prevState.isOn
    }));
  }

  render() {
    return (
      <button onClick={this.handleClick}>
        {this.state.isOn ? '켜짐' : '꺼짐'}
      </button>
    );
  }
}
```

## 함수형 컴포넌트에서의 이벤트

```
// 함수형 컴포넌트에서 이벤트 처리
function Form() {
  const [name, setName] = useState("");

  const handleChange = (event) => {
    setName(event.target.value);
  };

  const handleSubmit = (event) => {
    event.preventDefault();
    alert(`이름: ${name}`);
  };

  return (
    <form onSubmit={handleSubmit}>
      <input
        type="text"
        value={name}
        onChange={handleChange}
      />
      <button type="submit">제출</button>
    </form>
  );
}
```

# React 리스트와 키

배열 데이터

React에서 배열 데이터를 렌더링하기 위한 기본 형태입니다.

map()으로 렌더링

JavaScript의 map() 함수를 사용하여 배열의 각 항목을 React 요소로 변환합니다.

고유 키 사용

각 항목을 구분하기 위해 고유한 키 값을 사용합니다. 객체의 ID가 가장 적합합니다.

키의 중요성

React가 어떤 항목이 변경, 추가 또는 제거되었는지 식별하는 데 도움을 줍니다. 배열 내 항목에 안정적인 ID를 제공해야 합니다.

```
const numbers = [1, 2, 3, 4, 5];
const people = [
  { id: 1, name: '홍길동', age: 25 },
  { id: 2, name: '김철수', age: 30 },
  { id: 3, name: '박영희', age: 28 }
];
```

```
function NumberList() {
  return (
    <ul>
      {numbers.map((number) =>
        <li key={number}>{number}</li>
      )}
    </ul>
  );
}
```

```
function PeopleList() {
  return (
    <ul>
      {people.map((person) =>
        <li key={person.id}>
          {person.name} ({person.age}세)
        </li>
      )}
    </ul>
  );
}
```

```
// 잘못된 예: 인덱스를 키로 사용
{items.map((item, index) =>
  <li key={index}>{item.text}</li>
)}
```

```
// 좋은 예: 고유 ID를 키로 사용
{items.map((item) =>
  <li key={item.id}>{item.text}</li>
)}
```

# React 조건부 렌더링

## if 조건문 사용

```
function UserGreeting(props) {  
  const isLoggedIn = props.isLoggedIn;  
  
  if (isLoggedIn) {  
    return <h1>환영합니다, 다시 오셨군요!</h1>;  
  } else {  
    return <h1>로그인해 주세요.</h1>;  
  }  
}
```

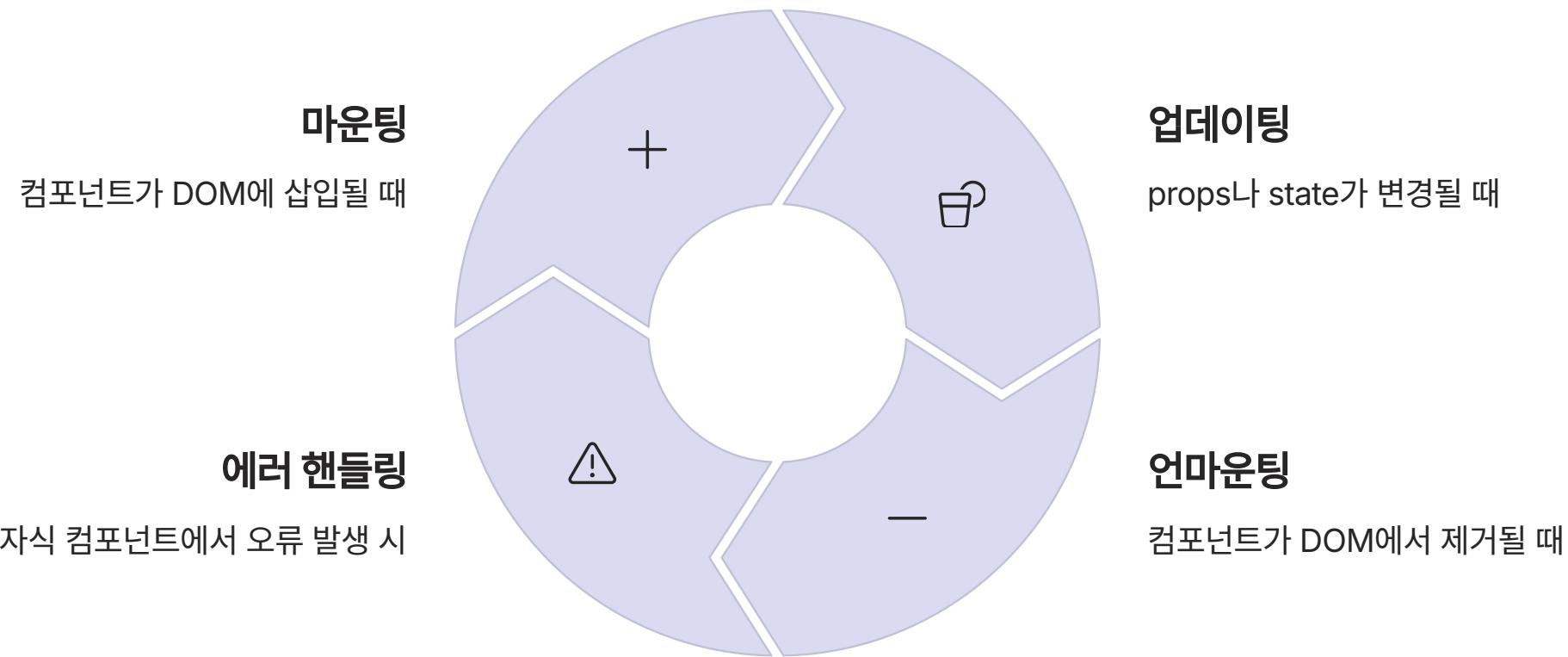
## 삼항 연산자 사용

```
function UserStatus(props) {  
  const isOnline = props.isOnline;  
  
  return (  
    <div>  
      {isOnline  
        ? <span className="online">온라인  
        </span>  
        : <span className="offline">오프라인  
      </span>  
    </div>  
  );  
}
```

## 논리 연산자 사용

```
function AdminPanel(props) {  
  const isAdmin = props.isAdmin;  
  
  return (  
    <div>  
      {isAdmin && (  
        <div className="admin-controls">  
          <h2>관리자 패널</h2>  
          <button>사용자 관리</button>  
          <button>설정</button>  
        </div>  
      )}  
    </div>  
  );  
}
```

# React 생명주기 메서드



# React Hooks

함수형 컴포넌트에서 상태 관리와 생명주기 기능을 사용할 수 있게 해주는 기능

훅 이름	사용 목적	주요 특징
useState	함수형 컴포넌트에서 상태 관리를 위한 훅	상태 변수와 상태 업데이트 함수를 반환함
useEffect	부수 효과를 처리하기 위한 훅	컴포넌트 렌더링 후 실행되며 정리(clean-up) 함수 지원
useContext	컨텍스트를 구독하기 위한 훅	컴포넌트 트리에서 데이터를 전역적으로 공유할 수 있음
useReducer	복잡한 상태 로직을 관리하기 위한 훅	리듀서 함수를 통해 상태 업데이트 로직을 분리할 수 있음
useRef	DOM 요소 참조 또는 값 저장을 위한 훅	렌더링에 영향을 주지 않고 값을 유지할 수 있음
useMemo	계산 결과를 메모이제이션하기 위한 훅	의존성이 변경될 때만 값을 재계산하여 성능 최적화
useCallback	함수를 메모이제이션하기 위한 훅	의존성이 변경될 때만 함수를 재생성하여 불필요한 렌더링 방지

# React 폼 처리

## 제어 컴포넌트

React 상태로 폼 요소의 값을 제어하는 방식입니다.

```
const [name, setName] = useState("");
const handleChange = (e) =>
  setName(e.target.value);
```

## 다양한 폼 요소

여러 유형의 입력 요소를 하나의 상태로 관리합니다.

```
const [inputs, setInputs] = useState({
  username: "",
  isSubscribed: false
});

const handleChange = (e) => {
  const { name, value, type, checked } =
    e.target;
  setInputs({...inputs,
    [name]: type === 'checkbox' ? checked :
    value
  });
};
```

## 폼 유효성 검사

사용자 입력을 검증하고 오류 메시지를 표시합니다.

```
const [email, setEmail] = useState("");
const [error, setError] = useState("");

// 간단한 이메일 검증
const isValid = /^[^\s@]+@[^\s@]+\.[^\s@]+$/ .test(email);
```

# React 라우팅 (React Router)

단계	설명	세부내용
1	설치 및 설정	라우터 패키지 설치 및 기본 설정
2	라우트 정의	URL 경로와 컴포넌트 연결
3	네비게이션 구현	링크 및 프로그래밍 방식 이동

```
// 패키지 설치  
// npm install react-router-dom  
  
// App.js  
import { BrowserRouter, Routes, Route, Link } from "react-router-dom";  
import Home from "./Home";  
import About from "./About";  
import Contact from "./Contact";  
  
function App() {  
  return (  
    <BrowserRouter>  
      <div>  
        <nav>  
          <ul>  
            <li><Link to="/">홈</Link></li>  
            <li><Link to="/about">소개</Link></li>  
            <li><Link to="/contact">연락처</Link></li>  
          </ul>  
        </nav>  
  
        <Routes>  
          <Route path="/" element={<Home />} />  
          <Route path="/about" element={<About />} />  
          <Route path="/contact" element={<Contact />} />  
        </Routes>  
      </div>  
    </BrowserRouter>  
  );  
}  
}
```

# React 상태 관리 라이브러리 (Redux)



## 스토어

1

애플리케이션의 상태를 저장

상태 저장소 - 앱의 모든 상태를 한 곳에 보관하는 중앙 창고



## 액션

⚡

상태 변경을 설명하는 객체

"이런 일이 발생했어요!"라는 선언 - 무슨 일이 일어났는지 알려주는 메시지

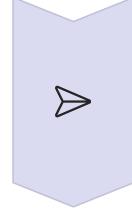


## 리듀서

▽

상태와 액션을 받아 새 상태 반환

"그럼 상태를 이렇게 바꿔야죠!"라고 반응하는 로직 - 액션에 따라 상태를 변경하는 함수



## 디스패치

➤

액션을 스토어에 보내는 함수

액션을 리듀서에게 전달하는 트리거 역할 - 이벤트 발생을 알리는 전달자

// 액션 타입 정의

```
const INCREMENT = 'INCREMENT';
const DECREMENT = 'DECREMENT';
```

// 액션 생성자

```
const increment = () => ({ type: INCREMENT });
const decrement = () => ({ type: DECREMENT });
```

// 초기 상태

```
const initialState = { count: 0 };
```

// 리듀서

```
const counterReducer = (state = initialState, action) => {
  switch (action.type) {
    case INCREMENT:
      return { count: state.count + 1 };
    case DECREMENT:
      return { count: state.count - 1 };
    default:
      return state;
  }
};
```

// 리액트 컴포넌트에서 사용

```
function Counter({ count, increment, decrement }) {
  return ( .....
```

# React 성능 최적화



## React.memo

컴포넌트를 메모이제이션하여 불필요한 리렌더링을 방지합니다.

```
const MemoizedComponent = React.memo(function
  MyComponent(props) {
    // 컴포넌트 렌더링 로직
    return <div>{props.value}</div>;
  });
}
```



## useMemo와 useCallback

계산 값과 함수를 메모이제이션하여 성능을 향상시킵니다.

```
// 비용이 큰 계산 메모이제이션
const memoizedValue = useMemo(() => {
  return computeExpensiveValue(a, b);
}, [a, b]);
```

```
// 함수 메모이제이션
const memoizedCallback = useCallback(() => {
  doSomething(a, b);
}, [a, b]);
```



## 가상 DOM 최적화

상태 및 생명주기 메서드를 효율적으로 사용하여 렌더링 최적화

```
// 불필요한 렌더링 방지
shouldComponentUpdate(nextProps, nextState) {
  return nextProps.id !== this.props.id;
}
```



## 리스트 최적화

안정적인 키를 사용하여 리스트 렌더링 성능 개선

```
// 고유한 키 사용
{items.map(item => (
  <Item key={item.id} {...item} />
))}
```

# React 서버와의 통신



## Fetch API 사용

브라우저 내장 API를 사용해 서버에서 데이터를 가져오고, useEffect와 useState를 활용하여 로딩 상태와 에러를 관리합니다.

## Axios 라이브러리 사용

HTTP 클라이언트 라이브러리인 Axios를 사용하여 더 간편하게 API 요청을 관리하고 응답 데이터를 처리합니다.

## 데이터 제출하기

폼 데이터를 관리하고 서버로 POST 요청을 보내어 사용자 입력을 제출하는 방법을 구현합니다.

# React 컴포넌트 스타일링

## 인라인 스타일

```
function Button() {  
  const buttonStyle = {  
    backgroundColor: 'blue',  
    color: 'white',  
    padding: '10px 15px',  
    borderRadius: '4px',  
    border: 'none',  
    cursor: 'pointer'  
  };  
  
  return (  
    <button style={buttonStyle}>  
      클릭하세요  
    </button>  
  );  
}
```

## CSS 모듈

```
// Button.module.css  
.button {  
  background-color: blue;  
  color: white;  
  padding: 10px 15px;  
  border-radius: 4px;  
  border: none;  
  cursor: pointer;  
}  
  
// Button.js  
import styles from './Button.module.css';  
  
function Button() {  
  return (  
    <button className={styles.button}>  
      클릭하세요  
    </button>  
  );  
}
```

## Styled Components

```
// npm install styled-components  
import styled from 'styled-components';  
  
const StyledButton = styled.button`  
  background-color: ${props => props.primary ? 'blue' : 'gray'};  
  color: white;  
  padding: 10px 15px;  
  border-radius: 4px;  
  border: none;  
  cursor: pointer;  
  
  &:hover {  
    opacity: 0.8;  
  }  
`;  
  
function Button({ primary, children }) {  
  return (  
    <StyledButton primary={primary}>  
      {children}  
    </StyledButton>  
  );  
}
```