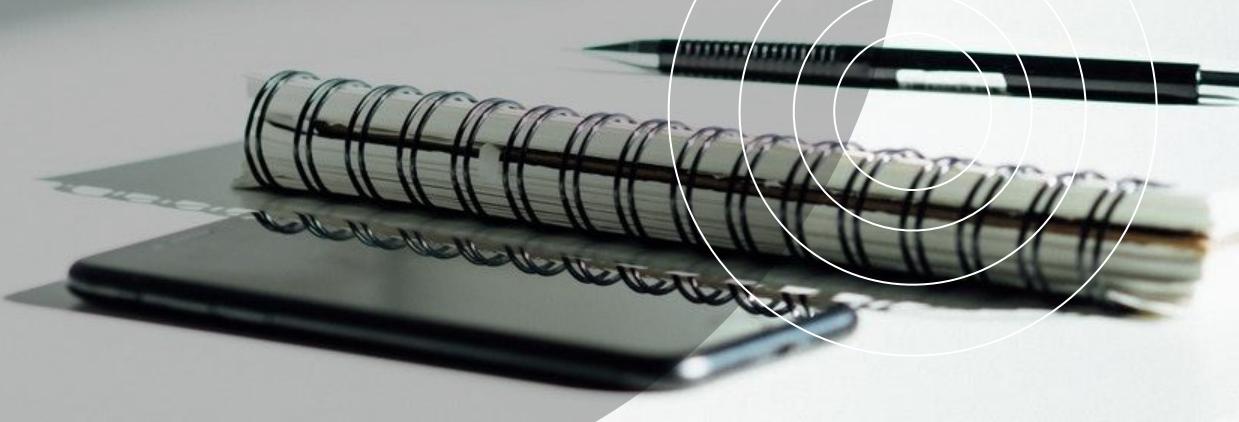


# 오픈소스 스튜디오

# Open Source Studio



12

# JavaScript Practice 1

# ECMAScript 6

---

- ◉ ES6 = ECMAScript 6
- ◉ The 6th version of ECMAScript in 2015
- ◉ React에서 ES6를 사용
- ◉ 특징
  - Classes
  - Arrow functions
  - let, const, var
  - Array Methods (map(),..)
  - Destructuring
  - Modules
  - Ternary Operator
  - Spread Operator

# JS Array

- Array는 여러가지 데이터를 하나의 변수로 처리
- Array는 여러 타입을 혼합하여 생성 가능
- 주요 Method

- array.sort()
- array.length
- array.forEach()
- array.push(new\_value)

- Array와 Object의 차이

- Numbered indexes

```
...
<script>
  const fruits1 = ["Apple", "Banana", "Grape"];

  const fruits2 = [];
  fruits2[0] = "Apple";
  fruits2[1] = "Banana";
  fruits2[2] = "Grape";

  const fruit3 = new Array("Apple", "Banana",
"Grape");
</script>
</head>
<body>

</body>
</html>
```

# JS Destructuring

- 구조분해할당 : 객체나 배열의 속성을 변수로 할당하는 방식

- Array Destructuring

- array 요소의 위치에 따라 할당

- Object Destructuring

- Object property 이름과 동일하게 분해
- Alias를 이용하여 변경가능
- 기본값 설정 가능

```
const fruits = ["Bananas", "Oranges", "Apples", "Mangos"];  
  
// Destructuring  
let [fruit1, fruit2] = fruits;  
let [fruit1,,,fruit2] = fruits;  
let {[0]:fruit1 ,[1]:fruit2} = fruits;  
const [a,b, ...rest] = numbers
```

```
const person = {  
    firstName: "John",  
    lastName: "Doe",  
    age: 50  
};  
  
// Destructuring  
let {lastName, firstName} = person;  
  
// Default value  
let {firstName, lastName, country = "US"} = person;  
  
// Property alias  
let {lastName : name} = person;
```

# JSON

---

## ◉ JSON (JavaScript Object Notation)

- 데이터를 저장하고 전송하는 텍스트 형식
- 대부분 프로그래밍 언어에서 JSON 읽고 생성하는 API 제공

## ◉ 특징

- key와 value로 구성
- 쉼표(,)로 데이터 구분
- 중괄호는 객체 보관 : {name : “jerry”, age : 20}
- 대괄호는 배열 보관 : [“apple”, “orange”, “kiwi” ]

```
// javascript 객체 예시
{firstName: "John", lastName: "Doe",
age: 50 }

// json 예시
{
    "firstName": "John",
    "lastName": "Doe",
    "age": 50
}
```

# JSON vs XML

- ◎ JSON과 XML(eXtensible Markup Lanaguage) 데이터 표현 형식
  - 3개의 employee 배열 데이터 표현

XML 표현

```
<employees>
  <employee>
    <firstName>John</firstName> <lastName>Doe</lastName>
  </employee>
  <employee>
    <firstName>Anna</firstName> <lastName>Smith</lastName>
  </employee>
  <employee>
    <firstName>Peter</firstName> <lastName>Jones</lastName>
  </employee>
</employees>
```

JSON 표현

```
{"employees": [
  { "firstName": "John", "lastName": "Doe" },
  { "firstName": "Anna", "lastName": "Smith" },
  { "firstName": "Peter", "lastName": "Jones" }
]}
```

# JS JSON

## ◎ JS에서 제공하는 함수

- JSON.parse() : JSON 문자열을 Javascript 객체로 변환
- JSON.stringify() : 객체를 JSON 문자열로 변환

### JS-JSON Test

JSON.stringify()

[object Object]

JSON.parse()

```
// JS 객체
const user = { name : "jerry" , age : 20};

function toJson() {
    document.getElementById("demo").innerHTML = JSON.stringify(user);
}

function toJS() {
    let jstr = document.getElementById("demo").innerHTML;
    let obj = JSON.parse(jstr);
    document.getElementById("user").innerHTML = obj.name + "(" + obj.age + ")";
}
```

# JS Object

- **Object**

- Properties + function

- **Property**

- key + value

```
<script>
    // Create an Object
    const car = {};

    // Add Properties
    car.name = "Genesis";
    car.year = "2024";

    let myCar = {
        name: 'Ford',
        year: 2015,
        print : function(){
            return "Car Info [" + this.name + "] - " + this.year;
        }
    };

    document.write(myCar.print());

</script>
```

# Classes

- ES6 introduced classes
- keyword
  - class
  - constructor()
  - super, this
  - extends

```
<!DOCTYPE html>
<html>
...
<script>
class Car {
  constructor(name) {
    this.brand = name;
  }

  present() { return 'I have a ' + this.brand; }
}

class Model extends Car {
  constructor(name, model) {
    super(name);
    this.model = model;
  }

  show() {
    return this.present() + ', it is a ' + this.model
  }
}

const mycar = new Model("Ford", "Mustang");
document.write(mycar.show());
</script>

</body>
</html>
```

# JS Arrow function

- ◉ ES6에서 소개
- ◉ Array function => Lambda expression
- ◉ 함수를 간단하게 표현

```
let hello = function () {  
    return "Hello World!";  
}  
  
let hello2 = () => {return "Hello World!"}
```

```
//parameters  
const func1 = (a, b) => a + b;  
  
//multiple lines  
const func2 = (a) => {  
    let b = 10;  
    return a + b;  
};  
  
// one parameter  
const func3 = a => a * 50;  
  
// no parameter  
const func4 = () => {};
```

# JS Hoisting

- JS인터프리터가 코드 실행 전 함수, 변수, 클래스 등을 유효범위 안에서 최상단으로 끌어올리는 것처럼 보이는 현상
  - Value hoisting, Function Hoisting
- hoisting으로 인해 로직 변경 함수는 가능하면 최상단에 작성하는 것이 좋음
- TDZ(Temporal Dead Zone)
  - let, const로 선언한 변수는 hoisting되지만, 변수 접근이 안됨!

```
<script>
    hoisting_test();

    function hoisting_test() {
        document.writeln("Hoisting!<br>");
    }

    hoisting_test();
</script>
```

```
<script>
    console.log(name);
    var name = "Jerry";
    console.log(name);

</script>
```

```
<script>
    console.log(name);
    let name = "Jerry";
    console.log(name);

</script>
```

# Window Object(BOM)

## ◎ Window Object

- JS window 객체는 웹 브라우저 창이나 탭을 대표하는 객체
- 웹 브라우저 환경에서 window 객체는 전역 객체 작용
- 웹 페이지의 내용, 웹 브라우저 자체와 관련된 메서드, 속성, 이벤트를 접근하고 조작할 수 있는 인터페이스 제공
- 브라우저에서 제공하는 객체
- window 객체로 브라우저의 창에 대한 정보를 알고, 제어 가능

```
> window
<- Window {0: Window, window: Window, self: Window, docu
  document, name: "", Location: Location, ...} ⓘ
  > 0: Window {window: Window, self: Window, document:
  > JSCompiler_renameProperty: f (t,e)
  > alert: f alert()
  > atob: f atob()
  > blur: f blur()
  > btoa: f btoa()
  > caches: CacheStorage {}
  > cancelAnimationFrame: f cancelAnimationFrame()
  > cancelIdleCallback: f cancelIdleCallback()
  > captureEvents: f captureEvents()
  > chrome: {metricsPrivate: {...}, loadTimes: f, csi: f,
  > clearInterval: f clearInterval()
  > clearTimeout: f clearTimeout()
  > clientInformation: Navigator {vendorSub: "", produc
  > close: f close()
  > closed: false
  > confirm: f confirm()
  > cookieStore: CookieStore {onchange: null}
  > createImageBitmap: f createImageBitmap()
  > credentialless: false
  > crossOriginIsolated: false
  > crypto: Crypto {subtle: SubtleCrypto}
  > customElements: CustomElementRegistry {}
  > devicePixelRatio: 1.5
```

# Window Object 특성

---

## ◎ 전역 변수와 함수

- 웹 페이지 내 전역 범위에 선언된 변수와 함수는 기본적으로 window 객체의 속성 및 메서드로 접근 가능

## ◎ 타이머 메서드

- setTimeout(), setInterval(), clearTimeout(), clearInterval()와 같은 타이머 관련 메서드를 제공

## ◎ 브라우저 제어

- 웹 페이지의 경고 창, 확인 창, 프롬프트 박스 등을 표시할 수 있는 메서드 (alert(), confirm(), prompt())를 포함하고 있음

## ◎ 문서 객체 모델(DOM) 접근

- window 객체를 통해 document 객체를 접근 가능
- document 객체는 웹 페이지의 내용을 조작하거나 조회할 수 있는 인터페이스를 제공

# Window 객체의 주요 특성과 기능

---

## ● 브라우저 정보

- window 객체는 현재 브라우저와 관련된 정보를 제공하는 navigator 객체에 접근 가능

## ● 브라우저 창 제어

- window 객체를 통해 현재 브라우저 창의 크기를 조절하거나 새 창을 열거나 닫는 등의 작업

## ● 이벤트

- 브라우저 창이나 웹 페이지 전체에 발생하는 이벤트, 예를 들면 resize, scroll 등을 감지하고 반응URL 조작
- window.location 객체를 통해 현재 페이지의 URL을 조회하거나 변경

## ● 이력 관리

- window.history 객체를 통해 브라우저의 세션 히스토리를 조작하거나 조회

# JS Browser BOM(1/2)

---

## ◉ BOM(Browser Object Model)

- window Object – Global functions, Global variable
  - open(), close(), moveTo(), resizeTo()
- screen Object
  - screen.width / screen.height
  - screen.availWidth / screen.availHeight
  - screen.colorDepth / screen.pixelDepth
- location Object
  - href / hostname / pathname / protocol

go Naver.com

back()

forward()

Navigator App

# JS Browser BOM(2/2)

---

- BOM(Browser Object Model)

- History Object

- `history.back()` / `history.forward()`

- Navigator Object

- Popup Alert

- `alert()`, `confirm()`, `prompt()`

- Timing

- `setTimeout(func, milliseconds)`, `setInterval(func, milliseconds)`
    - `clearTimeout(time_var)`, `clearInterval(time_var)`

- Cookies

- `document.cookie`

```
document.cookie = "username=John Doe;  
expires=Thu, 18 Dec 2013 12:00:00 UTC";
```

13

# JavaScript Practice 2

# TypeScript

---

- 2012년, MS(Microsoft)에 의해 개발된 프로그래밍 언어
- 안정적인 개발과 높은 수준의 코드 품질 유지
- Superset of JavaScript
  - Javascript 엔진 사용 및 Javascript로 컴파일하고 실행
- Key Features
  - Static typing / Type inference / Classes and Interfaces
  - Modules – ES6 module syntax
  - Compatibility : browsers and server-side platforms, JavaScript environments
  - Tooling and IDE Support
  - Enhanced Error Checking

# TypeScript vs JavaScript

- 정적 타입/ 동적 타입

- TypeScript는 변수의 타입을 **명시적으로** 지정하여 사용

- 컴파일 언어(Typescript) / 인터프리터 언어(Javascript)

- npm으로 typescript 설치

- .ts 파일을 생성하고, 컴파일 하면 동일한 기능을 수행하는 js 생성

- 학습의 차이

- TypeScript는 변수의 자료형의 이해, 모듈설치, 컴파일 등.

- 대규모 어플리케이션에 효과적!



# Code Example

```
class Greeter {  
  constructor(message) {  
    this.greeting = message;  
  }  
  
  greet() {  
    return `Hello, ${this.greeting}`;  
  }  
}  
  
let greeter = new Greeter("World");  
  
console.log(greeter.greet());
```

javascript

```
class Greeter {  
  greeting: string;  
  
  constructor(message: string) {  
    this.greeting = message;  
  }  
  
  greet(): string {  
    return `Hello, ${this.greeting}`;  
  }  
}  
  
let greeter = new Greeter("World");  
  
console.log(greeter.greet());
```

typescript

# JQuery

- JavaScript Library
- JavaScript 프로그래밍을 간단하게 표현
- CDN or 다운로드하여 사용

If you click on me, I will disappear.

Click me ?

Click me ??

Click me ?

Click me ??

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.1/jquery.min.js"></script>
```

```
<script>
  $(document).ready(function () {
    $("#p1").click(function () {
      $(this).hide();
    });

    $("#p2").click(function () {
      $(this).css("color", "red");
    });

    $("#p3").click(function () {
      $("#p2").animate({ left: "250px" });
    });
  });
</script>
```

# JS AJAX

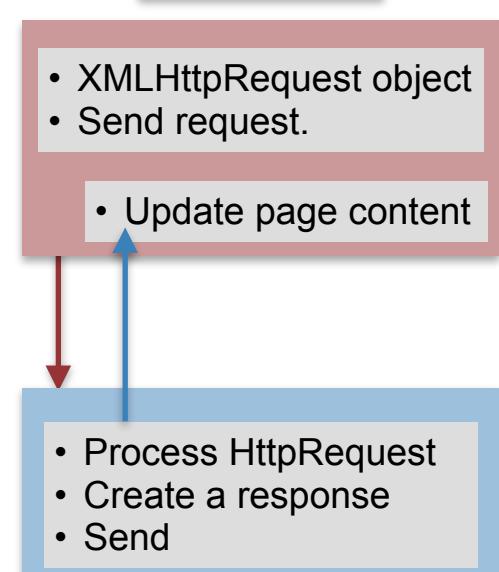
## ● AJAX를 사용하여 데이터 관리

- AJAX(Asynchronous JavaScript and XML)
- XMLHttpRequest Object
  - 브라우저와 웹 서버간에 통신을 해서 데이터를 가져올 수 있는 API
  - 동기/비동기 가능

## ● REST API(Representational State Transfer)

- REST 기반 API : Resource(URI), Method, Message
- HTTP Method
  - GET 리소스 요청 / POST 리소스 생성
  - PUT 리소스 수정 / DELETE 리소스 삭제

Web Browser



Server

# JS AJAX

---

## ◎ Synchronus vs Asynchronous

- 동기식(Synchronus)

- 여러개의 작업을 순차적으로 진행하는 것.
- 클라이언트가 서버에게 요청하고, 응답이 올 때까지 기다리는 상황

- 비동기식(Asynchronous)

- 여러 작업이 순차적으로 진행되지 않음
- 클라이언트가 서버의 응답을 기다리지 않고 다른 작업 수행

- 가능한 구현방법

- Javascript, JQuery, fetch API, AXIOS

# JS AJAX

---

## ◎ XMLHttpRequest Object (브라우저에 내장된 객체)

- XMLHttpRequest 객체 생성
- Callback function 정의
- XMLHttpRequest 객체 open(*method, url, async*)
- 서버에 Request 전송 send()

```
const xhttp = new XMLHttpRequest();
xhttp.onload = function() {
    document.getElementById("demo").innerHTML = this.responseText;
}
xhttp.open("GET", "demo_get.php");
xhttp.send();
```

# AJAX 실습(1/5)

## ◉ json-server 설치 : REST API Mock Server 구축

- npm install -g json-server
- data.json 파일 생성
- json-server --watch data.json

## ◉ Json-server 구동

```
lab % json-server --watch data.json --port 3001
```

data.json

## ◉ 실습연습

- GET - 데이터 리스트 가져오기
- POST - 새 데이터 추가
  - 새 데이터는 data.json 파일에 추가됨
- PUT / DELETE

```
{
  "students": [
    { "id": 1, "name": "Jerry", "age": 20 },
    { "id": 2, "name": "Tom", "age": 18 }
  ],
  "courses": [
    { "id": 1, "cname": "Open source studio", "credit": 3 },
    { "id": 2, "cname": "Programming studio", "credit": 2 }
  ]
}
```

# AJAX 실습(2/5)

## ● 실습연습

- HTML페이지 생성
- 이벤트 추가(addEventListener())이용
- GET Method - 데이터 가져오기

학생데이터 가져오기

name

age

학생데이터 추가

```
window.onload = function () {
    let btnStu = document.getElementById("btnStu");
    let btnAdd = document.getElementById("btnAdd");
    btnStu.addEventListener("click", getStudents);
    btnAdd.addEventListener("click", postData);
}

function getStudents() {
    let contents = document.getElementById("contents");
    const xhr = new XMLHttpRequest();

    xhr.open("GET", "http://localhost:3000/students");
    xhr.setRequestHeader("content-type", "application/json");
    xhr.send();

    xhr.onload = () => {
        if (xhr.status === 200) {
            const res = JSON.parse(xhr.response);
            contents.innerHTML = makeList(res);
        } else {
            console.log(xhr.status, xhr.statusText);
        }
    }
}
```

# AJAX 실습(3/5)

## ● 실습연습

- JSON 데이터를 parsing
- 데이터기반 동적 HTML 생성

학생데이터 가져오기

학생데이터 추가

- Jerry (20)
- Tom (18)

```
...
function getStudents() {
    ...

    xhr.onload = () => {
        if (xhr.status === 200) {
            const res = JSON.parse(xhr.response);
            contents.innerHTML = makeList(res);
        } else {
            console.log(xhr.status, xhr.statusText);
        }
    }
}

function makeList(data) {
    let str = "<ul>";
    for (key in data) {
        //console.log("Name :" + data[key].name + " , age :" + data[key].age);
        str += "<li> " + data[key].name + " (" + data[key].age + ")</li>";

    }
    str += "</ul>";
    return str;
}
```

# AJAX 실습(4/5)

## ● 실습연습

- POST를 이용한 데이터 추가
- HTML 페이지 수정
  - 데이터 입력 용 입력요소 추가
  - `JSON.stringify()` 함수
    - JS객체나 데이터를 JSON 문자열변환

학생데이터 가져오기

John	18	학생데이터 추가
------	----	----------

↓

name	age	학생데이터 추가
------	-----	----------

- Jerry (20)
- Tom (18)
- John (18)

```
function postData() {  
  
    let contents = document.getElementById("contents");  
    let name = document.getElementById("name");  
    let age = document.getElementById("age");  
    const xhr = new XMLHttpRequest();  
    xhr.open("POST", "http://localhost:3000/students");  
    xhr.setRequestHeader("content-type", "application/json;  
charset=UTF-8")  
    const data = { name: name.value, age: age.value };  
  
    xhr.send(JSON.stringify(data));  
    xhr.onload = () => {  
        if (xhr.status === 201) {  
            name.value = "";  
            age.value = "";  
            const res = JSON.parse(xhr.response);  
            getStudents();  
        } else {  
            console.log(xhr.status, xhr.statusText);  
        }  
    }  
}
```

# AJAX 실습(4/5)

## ● 실습연습

- PUT - 데이터 수정

- <http://localhost:3000/students/2>

- 수정하기 단계

- 기존데이터 확인
  - 데이터 수정
  - ajax를 통해 전송
  - 목록에 데이터 업데이트

```
function updateData(id) {
  const xhr = new XMLHttpRequest();
  xhr.open("PUT", "http://localhost:3000/students/" + id);
  xhr.setRequestHeader("content-type", "application/json; charset=UTF-8");
  const data = { name: "Sally", age: 10 };
  xhr.send(JSON.stringify(data));

  xhr.onload = () => {
    if (xhr.status === 200) {
      const res = JSON.parse(xhr.response);
      console.log(res);
      getStudents();
    } else {
      console.log(xhr.status, xhr.statusText);
    }
  }
}
```

14

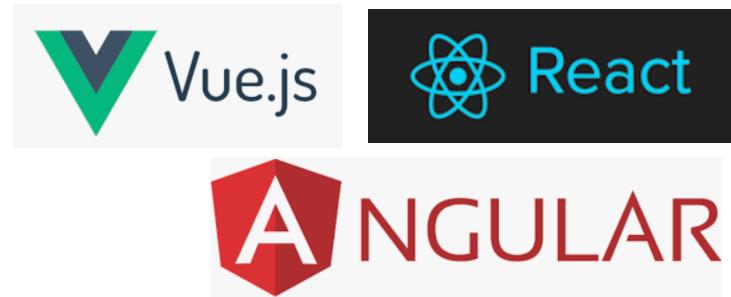
# React Basic

# JS Framework

---

## ● 프레임워크

- 코드작성이 효율적이며 이미 검증된 코드 사용
- 안정적인 유지보수
- 학습량이 많으며, 자유롭지 못한 제약이 있는 개발환경

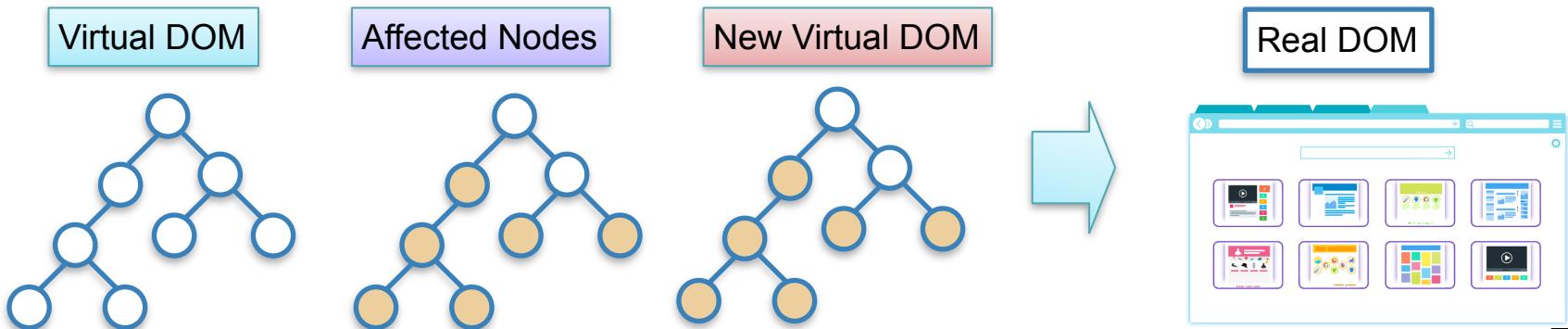


## ● JS Framework or Library

- Vue.js : 학습이 비교적 쉬움, 기존 프로젝트 적용하기 수월함, 속도가 빠름, 소규모
  - Gitlab, Adobe, Netflix 등
- React : 거대 사용자 커뮤니티, 페이스북의 지원, 다양한 레퍼런스 및 확장 라이브러리, 대규모
  - Facebook, MS, Airbnb 등
- Angular : 대규모 프로젝트, 완전한 프레임워크로 생성, 테스트, 빌드, 배포의 모든 기능 제공
  - 구글, 유튜브, 텔레그램, 나이키 등

# React

- ◉ 현재 버전 V18.2.0 (June 14, 2022)
- ◉ front-end Javascript framework
- ◉ Facebook에 의해 개발된 JavaScript library
- ◉ 메모리에 Virtual DOM 생성/변경 후 Real DOM(browser DOM)에 적용





# React 개발 환경 설정

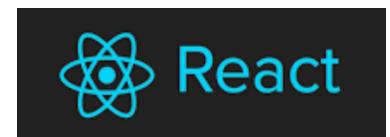
## ◉ Node.js 설치 – Javascript Runtime

- OS 버전에 맞게 LTS 버전 설치(npm, npx 설치됨)
  - **node -v**



## ◉ Yarn 설치 – Package manager module

- npm보다 빠르고 안정적



## ◉ create-react-app 설치

- npm 으로 설치
  - **npm install -g create-react-app**
- yarn 으로 설치
  - **yarn global add create-react-app**





# React 첫 프로젝트

## ● React 프로젝트 생성

- `create-react-app new-project-name`
- `npx create-react-app new-project-name`

## ● 새 프로젝트로 이동

- `cd new-project-name`

## ● 새 프로젝트 실행

- `npm start / yarn start`

## ● vscode를 이용한 개발

- Extension 설치
- git으로 버전관리
- github로 push

```
Compiled successfully!
```

```
You can now view test-app in the browser.
```

```
Local:          http://localhost:3000  
On Your Network:  http://172.17.194.183:3000
```

```
Note that the development build is not optimized.  
To create a production build, use npm run build.
```

```
webpack compiled successfully
```



### ES7+ React/Redux/React-Native snippets

dsznajder | 12,378,951 installs | ★★★★★ (78) | Free

Extensions for React, React-Native and Redux in JS/TS with ES7+ syntax. Customizable. Built-in integration with prettier.

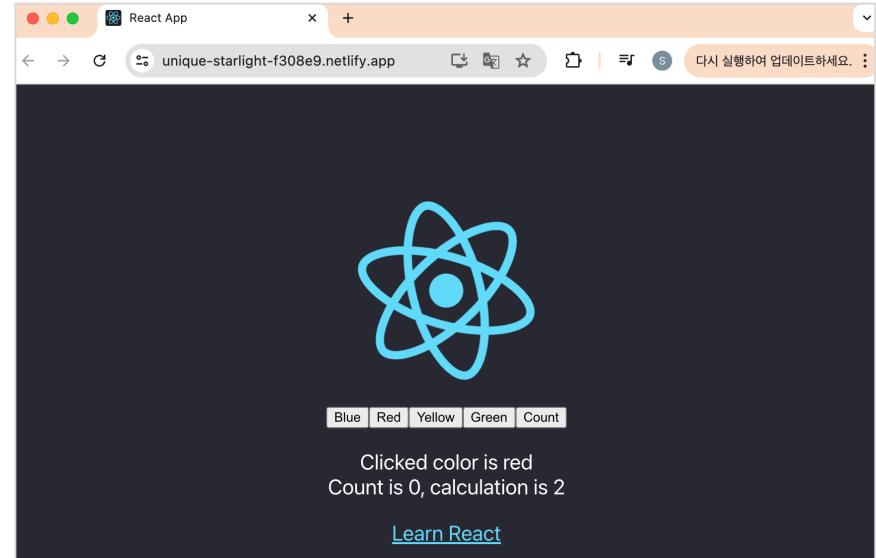
rce rcc 등을 입력하면 자동완성



# React 프로젝트 배포

## ◎ Netlify 이용한 배포

- 무료용량, SSL인증서 자동 제공, CI/CD 구축 가능
- 회원가입 / 로그인 / 팀 생성
- 새로운 사이트 추가
  - Github의 Repository 연동
- 배포



# React 프로젝트 배포2

---

## ◎ Github Classroom Assignment Repo를 이용한 배포

1. Assignment github repo 클론 후 폴더 이동
2. 현재 폴더에 새로운 React package 생성
3. 모든 파일 버전 기록 후 push
4. 내 github에 새로운 repo 생성
5. [My Repo] repo에 push
6. [My Repo]에 netlify 연결

```
react % git clone my_assignment_github_repo_url
react % cd my_assignment_github_repo_url
..repo_url % npx create-react-app ./
..repo_url % npm start
..repo_url % git remote -v
..repo_url % git add .
..repo_url % git commit -m "create react project"
..repo_url % git push origin main
..repo_url % git remote add origin my_github_new_repo_url
..repo_url % git push myorigin main
```

# Hello React Project

- ◎ nodejs / Create-React-App 설치 (windows – powershell)

```
npm install -g create-react-app
```

- ◎ npm / npx 를 이용하여 새 프로젝트 만들기
- ◎ 새 프로젝트 실행 (<http://localhost:3000/>)

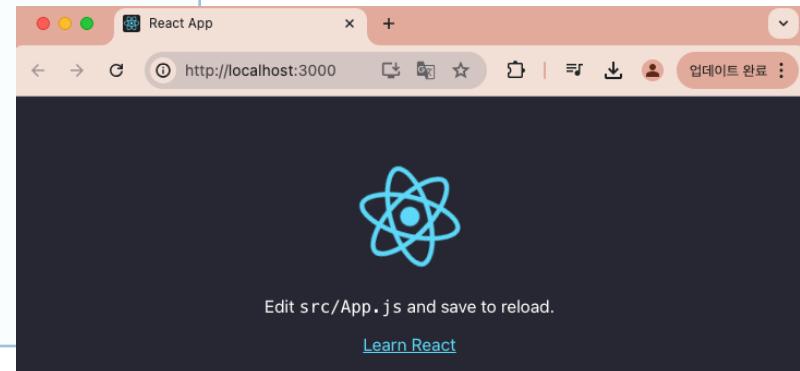
```
Macmini react % npx create-react-app hello-react-app
```

```
...  
We suggest that you begin by typing:
```

```
cd hello-react-app  
npm start
```

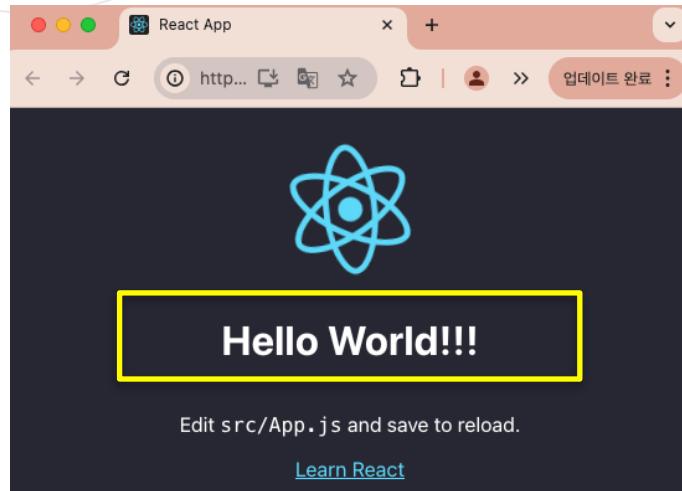
```
Happy hacking!
```

```
Macmini react % cd hello-react-app  
Macmini react % npm start
```



# 간단한 Component 추가

- ◉ Hello.css 새 파일 추가
- ◉ Hello.js 새 파일 추가
- ◉ App.js에 Hello component 추가



```
# Hello.css U JS Hello.js U X
src > JS Hello.js > [o] default
1 import './Hello.css';
2
3 function Hello(props){
4   return (
5     <h1>Hello World!!!</h1>
6   );
7 }
8
9 export default Hello;
```

```
# Hello.css U JS Hello.js U JS App.js M X
src > JS App.js > [o] App
1 import logo from './logo.svg',
2 import './App.css';
3 import Hello from './Hello.js';
4
5 function App() {
6   return (
7     <div className="App">
8       <header className="App-header">
9         <img src={logo} className="App-logo" alt="React logo" />
10        <Hello />
11       <p>
```

15

# React Practice

# MPA vs SPA

---

## ◎ MPA(Multiple Page Application) 방식

- 필요한 다수의 페이지를 생성
- 전체 페이지를 로딩하는 속도가 느려지는 이슈가 있음
- AJAX(Asynchronous Javascript And XML)를 이용하여 MPA를 개선하고자 함
- SSR(Server-Side Rendering) 방식

## ◎ SPA(Single Page Application) 방식

- 한개의 페이지로 구성된 application
- 전체 페이지를 서버로부터 가져오지 않고 동적으로 변경부분만 다시 가져옴
- UX(User eXperience) 향상이 핵심 가치, 애플리케이션의 속도 향상
- CSR(Client-Side Rendering) 방식 사용

# SPA 의 특징

---

## ◎ SPA의 장점

- 속도 및 응답 시간 향상 (mobile first)
- 모바일 친화적이며, backend api를 이용하여 웹이나 앱 UI 개발
- Frontend 렌더링을 위한 코드 생성으로 개발 간소화
- Local Storage 캐시

## ◎ SPA의 단점

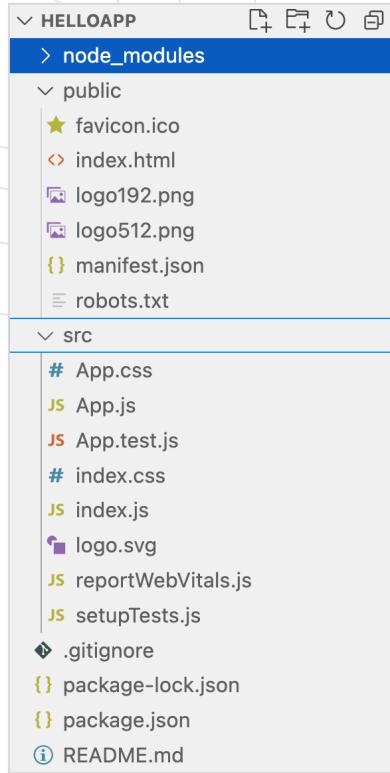
- 초기 구동속도가 상대적으로 느림
- SEO(Search Engine Optimization) 이슈
  - JS를 읽지 못하는 검색엔진은 크롤링 어려움
- XSS(Cross Site Scripting) 에 취약

# React Library

---

- React는 라이브러리이고, 필요한 것들은 모듈을 설치해 사용해야 함.
- Webpack과 Babel
  - Webpack : JavaScript Module Bundler
    - 여러개의 모듈(html, css, JS 등)을 하나의 JS파일로 묶어주는 Module Bundler
  - Babel : JavaScript Compiler
    - 최신 버전의 JS를 구형 버전의 JS로 변환해줌.
- 상태관리: redux, recoil, mobx 등
- Build : webpack, npm 등
- 라우팅(Routing) : react-router-dom
- 테스트 : Eslint, Mocha 등

# React Project 구조



## ◎ node\_modules

- 모든 패키지 소스코드가 있는 폴더

## ◎ Public

- index.html 와 같은 정적 파일이 포함되는 위치

## ◎ .gitignore

## ◎ package.json

- 기본 패키지 및 추가 라이브러리 패키지 정보 관리 파일

## ◎ README.md

- 현재 프로젝트 소개 페이지

## ◎ 부가 폴더

- assets, components, config, constants, hooks, pages, services, styles

# React Rendering(1/2)

---

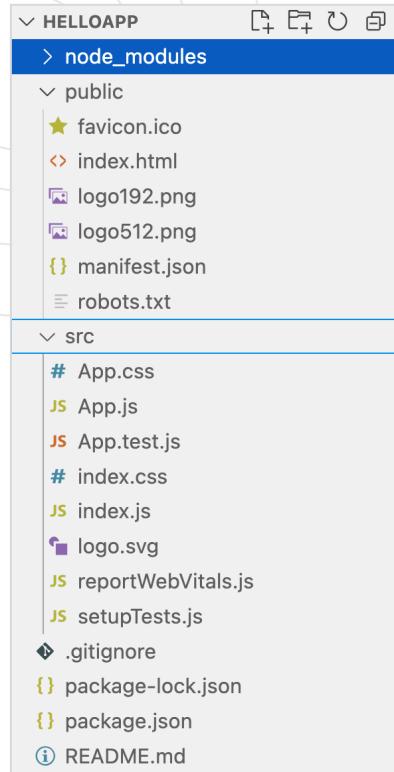
## ● Rendering

- Browser Rendering : HTML과 CSS를 기반으로 웹 페이지에 필요한 UI를 그려냄
- React Rendering : DOM Tree를 만드는 과정(Props와 State로 UI 구성)

## ● Rendering 발생 구분

- First Render
- Re-renders
  - Class Component : setState(), forceUpdate() 실행하는 경우
  - Function Component
    - useState()의 setter 실행하는 경우
    - useReducer()의 dispatch가 실행되는 경우
    - 부모 컴포넌트가 렌더링 되는 경우 등.

# React Rendering(2/2)



index.html

```
<noscript>You need to enable JavaScript to run this app.</noscript>
<div id="root"></div>
```

index.js

```
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

App.js



# React 프로젝트 배포2

## ◎ Github Classroom Assignment Repo를 이용한 배포

1. Assignment github repo 클론 후 폴더 이동
2. 현재 폴더에 새로운 React package 생성
3. 모든 파일 버전 기록 후 push
4. 내 github에 새로운 repo 생성
5. [My Repo] repo에 push
6. [My Repo]에 netlify 연결
7. **github\_url에 대문자가 있는 경우 Project name 오류**  
**\* 폴더이름 변경**

```
react % git clone my_assignment_github_repo_url
react % cd my_assignment_github_repo_url
..repo_url % npx create-react-app ./
..repo_url % npm start
..repo_url % git remote -v
..repo_url % git add .
..repo_url % git commit -m "create react project"
..repo_url % git push origin main
..repo_url % git remote add origin my_github_new_repo_url
..repo_url % git push myorigin main
```

# JSX란?

- JSX(Javascript XML)
- ES6 지원
- JSX는 React에서 사용되며,  
브라우저에서 실행하기전 JS  
로 변환됨(Babel)
- JSX는 한 파일에 JS와 HTML  
을 동시에 작성 가능

```
function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />

        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </a>
      </header>
    </div>
  );
}
```

# JSX 문법

---

- 반드시 하나의 부모 요소가 있어야 함
  - DOM Tree 특징
  - <></> 로 감싸도 됨.
- {JS표현식} 사용 가능
- 다수 라인의 HTML tag는 () 사이에 추가
- html이나 css attribute 이름은 camelCase 사용
  - background-color => backgroundColor
- class는 React에서 예약어이므로, class 속성은 className으로 변경
- 주석 : { /\* comment \*/ } 이나 // 표기
- if문은 HTML과 혼용하여 사용할 수 없음(삼항 연산자 가능)

# JSX 문법 예제

```
const myElement = <h1>I Love JSX!</h1>; // JSX
```

```
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(myElement);
```

```
//without JSX
const myElement = React.createElement('h1', {}, 'I do not use JSX!');

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(myElement);
```

```
const myElement = <h1>React is {5 + 5} times better with JSX</h1>;
```

```
const myElement = (
  <ul>
    <li>Apples</li>
    <li>Bananas</li>
    <li>Cherries</li>
  </ul>
);
```

```
const myElement = (
  <>
    <p>I am a paragraph.</p>
    <p>I am a paragraph too.</p>
  </>
);
```

```
const myElement = <input type="text" />;
```

```
const myElement = <h1 className="myclass">Hello World</h1>;
```

```
const x = 5;
let text = "Goodbye";
if (x < 10) {
  text = "Hello";
}
```

```
const myElement = <h1>{text}</h1>;
```

# JSX Styling

## ● Inline Styling

- 태그에 직접 styling
- `{}` 안에 작성

## ● CSS Stylesheets

## ● CSS Modules

```
const Header = () => {
  return (
    <>
      <h1 style={{color: "red"}}>Hello Style!</h1>
      <p>Add a little style!</p>
    </>
  );
}
```

```
const CSSSample = () => {
  const myStyle = {
    color: "white",
    backgroundColor: "DodgerBlue",
    padding: "10px",
    fontFamily: "Sans-Serif"
  };
  return (
    <>
      <h1 style={myStyle}>Hello Style!</h1>
      <p>Add a little style!</p>
    </>
  );
}
```

```
import './App.css';
```

# Without using JSX for Example 1

- JSX converts HTML tags into react elements.

```
// src/index.js
import React from 'react';
import ReactDOM from 'react-dom/client';

const myElement = React.createElement('h1', {}, 'I do not use JSX!');
const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(myElement);
```

# Coding JSX: Example 1

- JSX converts HTML tags into react elements.

```
// src/index.js
import React from 'react';
import ReactDOM from 'react-dom/client';

// write HTML directly within the JavaScript code
const myElement = <h1>This is HTML tags!!</h1>;
const root = ReactDOM.createRoot(document.getElementById('root'));

// the html tags <h1>...</h1> is converted into a react element and then pass to
// the render function.

root.render(myElement);
```

# Expressions in JSX

- With JSX you can write expressions inside curly braces {}.
- Execute the expression  $5 + 5$

```
// src/index.js
import React from 'react';
import ReactDOM from 'react-dom/client';

// write HTML directly within the JavaScript code
const myElement = <h1>React is {5 + 5} times better with JSX</h1>;
const root = ReactDOM.createRoot(document.getElementById('root'));

// the html tags <h1>...</h1> is converted into a react element and then pass to
// the render function.

root.render(myElement);
```

# Inserting a Large Block of HTML

- ◎ To write HTML on multiple lines, put the HTML inside parentheses:
  - Create a list with three list items:

```
// src/index.js
import React from 'react';
import ReactDOM from 'react-dom/client';

// write HTML directly within the JavaScript code
const myElement = (
  <ul>
    <li>Apples</li>
    <li>Bananas</li>
    <li>Cherries</li>
  </ul>
);
const root = ReactDOM.createRoot(document.getElementById('root'));

// the html tags <h1>...</h1> is converted into a react element and then pass to
// the render function.

root.render(myElement);
```

# Rule 1: Must be ONE top level element

- The HTML code must be wrapped in ONE top level element.

- Wrap two paragraphs inside one DIV element (You can use put a react fragment-empty tag-`<> </>`):

```
// src/index.js
import React from 'react';
import ReactDOM from 'react-dom/client';

// write HTML directly within the JavaScript code
const myElement = (
  <div>
    <p>I am a paragraph.</p>
    <p>I am a paragraph too.</p>
  </div>
);
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(myElement);
```

- JSX will throw an error if the HTML is not correct, or if the HTML misses a parent element.

# Rule 2: Elements must be CLOSED.

- ◉ JSX follows XML rules, and therefore HTML elements must be properly closed.
  - Close empty elements with />

```
// src/index.js
import React from 'react';
import ReactDOM from 'react-dom/client';

const myElement = <input type="text" />;
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(myElement);
```

- JSX will throw an error if the HTML is not properly closed.

# Rule 3: Attribute class = className

- The 'class' keyword is a reserved word in JavaScript, you are not allowed to use it in JSX. Instead, use 'className' for a attribute name in HTML.

```
// src/index.js
import React from 'react';
import ReactDOM from 'react-dom/client';

const myElement = <h1 className="myclass">Hello World</h1>;
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(myElement);
```

# Rule 4: Conditions – if statements (1)

- React supports if statements, but not inside JSX.
- To be able to use conditional statements in JSX, you should put the if statements outside of the JSX, or you could use a ternary expression instead:
  - Option 1: Write *if* statements outside of the JSX code:
    - Write "Hello" if x is less than 10, otherwise "Goodbye":

```
const x = 5;  
let text = "Goodbye";  
  
if (x < 10) {  
  text = "Hello";  
}  
  
const myElement = <h1>{text}</h1>;
```

# Rule 4: Conditions – if statements (2)

- Option 2: Use ternary expressions
  - Write "Hello" if x is less than 10, otherwise "Goodbye":

```
...
const x = 5;
const myElement = <h1>{ (x) < 10 ? "Hello" : "Goodbye" }</h1>;
...
```

- Note that in order to embed a JavaScript expression inside JSX, the JavaScript must be wrapped with curly braces, {}.

16

# React Components

# React Component

---

## ◉ Component?

- 독립적이고 재사용할 수 있는 요소
- JS의 함수와 동일한 목적을 제공하며 HTML을 반환
- Component이름의 첫 글자는 대문자로 시작

## ◉ Component 종류

- Class component
- Function component

# React Component

---

## ● 클래스형 컴포넌트(Class component)

- Legacy project, 이전 버전에서 다수 작업
- componentDidMount 등록 후 componentWillUnmount에서 해제
  - Lifecycle method 필요
- 코드 재사용성이 낮고 구성이 어려움.

## ● 함수형 컴포넌트(Function Component)

- 간단한 유형의 컴포넌트
- 간결하고 이해하기 쉬워 최근 많이 사용
- useState, useEffect, useContext 등의 React hook 사용(v16.8에서 hooks 도입)
  - Hooks : React state와 생명주기 기능(lifecycle features)을 연동

# React Component

## 함수형 컴포넌트(Function Component)

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

## 클래스형 컴포넌트(Class component)

```
class Welcome extends React.Component {  
    render() {  
        return <h1>Hello, {this.props.name}</h1>;  
    }  
}
```

# React Component 예시

- Component는 새 파일로 생성
- Component 내부에 Component를 생성할 수 있음
- 필요한 곳에서 import 하여 component 사용 가능

Arrow function

```
import './Hello.css';

function Hello(){
  return (
    <h1>안녕하세요 여러분 !!! </h1>
  );
}

export default Hello;
```

```
import './Hello.css';

const Hello = (props) => {
  return (
    <h1>안녕하세요 여러분!!! </h1>
  );
}

export default Hello;
```

# Props(Properties)

- Component에 Property를 설정하기 위해 props를 사용
- Component 태그를 사용할 때 attribute와 value 설정 가능
- 작업
  - Hello Component 생성
  - props parameter를 통해 속성명으로 사용
  - Tag에서 attr와 value 사용법과 동일

```
import logo from './logo.svg';
import './App.css';

function Hello(props){
  return (
    <h1>Hello {props.name}!!! </h1>
  );
}

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <Hello name="Jerry" />
      <p>
        Edit <code>src/App.js</code> and save to reload.
      </p>
    </header>
  );
}

export default App;
```

# Props – Pass data

- ◉ 값으로 전달
- ◉ 변수를 이용한 전달
- ◉ 객체로 전달

```
function Car(props) {
  return <h2>I am a {props.brand}!</h2>;
}

function Garage() {
  const carName= "Ford";

  return (
    <div>
      <h1>Who lives in my garage?</h1>
      <Car brand={carName} />
    </div>
  );
}
```

```
function Car(props) {
  return <h2>I am a {props.brand}!</h2>;
}

function Garage() {
  return (
    <div>
      <h1>Who lives in my garage?</h1>
      <Car brand="Ford" />
    </div>
  );
}
```

```
function Car(props) {
  return <h2>I am a {props.brand.name}!</h2>;
}

function Garage() {
  const carInfo= {name : "Ford", model:"Mustang"};

  return (
    <div>
      <h1>Who lives in my garage?</h1>
      <Car brand={carName} />
    </div>
  );
}
```

# Array.map()

- ◎ map 메서드는 array의 모든 요소에 대해 Callback 함수를 호출하고, 그 결과를 모아 새로운 배열을 반환함

```
const array1 = [1, 4, 9, 16];
// Pass a function to map
const map1 = array1.map((x) => x * 2);
console.log(map1);
// Expected output: Array [2, 8, 18, 32]
```

```
function Car(props) {
  return <li>I am a { props.brand }</li>;
}
function Garage() {
  const cars = ['Ford', 'BMW', 'Audi'];
  return (
    <div>
      <h1>Who lives in my garage?</h1>
      <ul>
        {cars.map((car) => <Car brand={car} />)}
      </ul>
    </div>
  );
}
```

# List와 Key

- ◉ 컴포넌트에서 list를 렌더링 할 때, key를 넣어야 한다는 경고 표시
- ◉ 리스트 아이템을 추가, 수정, 삭제 할 때 key를 통해 식별(unique id 사용)

```
function Car(props) {  
  return <li>I am a { props.brand }</li>;  
}  
  
function Garage() {  
  const cars = ['Ford', 'BMW', 'Audi'];  
  return (  
    <div>  
      <h1>Who lives in my garage?</h1>  
      <ul>  
        {cars.map((car) => <Car brand={car} />)}  
      </ul>  
    </div>  
  );  
}
```

✖ Warning: Each child in a list should have a unique "key" prop.  
Check the render method of 'Garage'. See [ht](#) more information.  
at Car (http://localhost:3000/main.2f73  
at Garage  
at header

```
function Car(props) {  
  return <li>I am a { props.brand }</li>;  
}  
  
function Garage() {  
  const cars = [  
    {id: 1, brand: 'Ford'},  
    {id: 2, brand: 'BMW'},  
    {id: 3, brand: 'Audi'}  
  ];  
  return (  
    <div>  
      <h1>Who lives in my garage?</h1>  
      <ul>  
        {cars.map((car) => <Car key={cars.id} brand={cars.brand} />)}  
      </ul>  
    </div>  
  );  
}
```

# React Class Components

---

- Before 16.8, Class components were the only way to track state and lifecycle on a React component.
- With the addition of Hooks, Function components are now almost equivalent to Class components and is recommended!
- However, you need to understand the old code using class components and you can still use class components.

# Create a Class Component

- ◉ Rule 1: When creating a React component, the component's name **MUST** start with an uppercase letter.
- ◉ Rule 2: **MUST include the *extends React.Component statement.***
  - This gives your component access to React.Component's functions.
  - Create a Class component called **Car**.

```
import React from 'react';
import ReactDOM from 'react-dom/client';

class Car extends React.Component {
  render() {
    return <h2>Hi, I am a Car!</h2>;
  }
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Car />);
```

# Component Constructor

- Use a constructor() function to initialize the component.
- Example

- The super() statement executes the parent component's constructor functionCreate a constructor function in the Car component, and add a color property:
- Use the color property in the render() function:

```
import React from 'react';
import ReactDOM from 'react-dom/client';
class Car extends React.Component {
  constructor() {
    super();
    this.state = {color: "red"};
  }
  render() {
    return <h2>I am a {this.state.color} Car!</h2>;
  }
}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Car />);
```

# Props

- Another way of handling component properties is by using props.
- Example
  - Use an attribute to pass a color to the Car component, and use it in the render() function:
  - If your component has a constructor function, the props should always be passed to the constructor and also to the React.Component via the super() method.

```
...
class Car extends React.Component {
  constructor(props) {
    super(props);
  }
  render() {
    return <h2>I am a {this.props.model}!</h2>;
  }
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Car model="Mustang"/>);
```

# Components in Components

- Use the Car component inside the Garage component:

```
...
class Car extends React.Component {
  render() {
    return <h2>I am a Car!</h2>;
  }
}

class Garage extends React.Component {
  render() {
    return (
      <div>
        <h1>Who lives in my Garage?</h1>
        <Car />
      </div>
    );
  }
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Garage />);
```

17

# React CRUD Open source Project

# React CRUD Open source project

- ◉ Open source project 검색 및 선택
  - [sample project](#)
- ◉ Rest API Server 구동(Open API, Json-Server 등)
- ◉ Clone Open Source Project
- ◉ Customizing

The screenshot shows a user interface for a React CRUD application. At the top, there is a dark header bar with the title "React CRUD" and three navigation links: "HOME", "CREATE USER", and "SHOW USER". Below the header is a table with a light gray background and a thin blue border. The table has columns for "ID", "Name", "Email", "Phone", and "Actions". There are four rows of data, each representing a user:

ID	Name	Email	Phone	Actions
1	Jerry	jerry@example.com	(054) 260-1234	
2	Tom	tom@sample.com	(031) 111-7410	
3	Sally	sally@test.com	(02) 123-7890	
4	홍길동	abc@hello.com	01012341234	

# Open Project + ClassRoom 연동

---

- ◉ 별칭정의

- Github\_Classroom\_Assignment\_repo => class\_repo
- Open\_source\_project\_repo => open\_repo

- ◉ open\_repo의 파일을 class\_repo로 복제

```
cp -rf ./open_repo/* class_repo
```

- ◉ class\_repo로 이동 (cd command)

- ◉ 다음 명령어 실행(dependency 설치)

```
npm i  
npm start
```

# REST API for react project

- ◎ <https://mockapi.io/> 서비스 이용

- JSON 데이터를 이용하여 Rest API 서비스 제공

- ◎ 생성과정

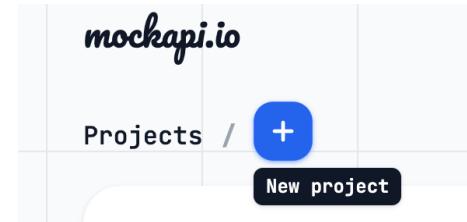
- New project 추가 (free plan : 1개 project) - sample

- New resource 추가 - user

<https://66fa0d90afc569e13a9a42b4.mockapi.io/sample/:endpoint>

- 새로운 Resource에 대한 schema 생성

- ◎ mockapi를 사용한 [sample service](#)



Schema

Define Resource schema, it will be used to generate mock data.

id

Object ID

email

Faker.js

internet.exampleEma...

name

Faker.js

name.fullName

phone

Faker.js

phone.number



18

# React Hooks

# React Events

---

- HTML DOM 요소에만 이벤트 지원

- click, change, mouseover, mousemove 등
- 사용자 생성 Component는 이벤트 지원 안함(props로 전달)

- React 이벤트 이름은 camel Case 사용(대소문자 중요)

- onClick, onChange

HTML

```
<button onclick="shoot()">Take the Shot!</button>
```

React

```
<button onClick={shoot}>Take the Shot!</button>
```

- e object(event object) 사용 가능

- 발생한 이벤트 관련 정보나 함수를 사용할 수 있음

# React Events – 예시

## ◎ Button에 onClick 이벤트 설정

// 이벤트 함수에 parameter가 없는 경우

```
function HelloBtn(){
  const message = () => {
    alert("Hello World!!!");
  }

  return (
    <button onClick={message}>Click Me</button>
  );
}

export default HelloBtn;
```

// 이벤트 함수에 parameter가 있는 경우

```
function HelloBtn2(){
  const message = (name) => {
    alert("Hello " + name + " !!!");
  }

  return (
    <button onClick={()=>{message("Sally")}}>Click Me</button>
  );
}

export default HelloBtn2;
```

# event object

## ◎ 이벤트 발생할 때 해당 요소에 발생하는 event object 사용

```
// 이벤트 함수에 event object 사용

function HelloBtn3(){
    const message = (name, event) => {
        alert("[" + event.target.id + "] Hello " + name + " !!!");
    }

    return (
        <button id="btn1" onClick={(e)=>{message("Sally",e)}}>Click Me</button>
    );
}

export default HelloBtn3;
```

# Adding Events

## ◎ Complete example

- Put the *shoot* function inside the *Football* component:

```
import React from 'react';
import ReactDOM from 'react-dom/client';

function Football() {
  const shoot = () => {
    alert("Great Shot!");
  }
  return (
    <button onClick={shoot}>Take the shot!</button>
  );
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Football />);
```

# Passing Arguments

- ◎ To pass an argument to an event handler, use an arrow function.
  - Send "Goal!" as a parameter to the shoot function, using arrow function:

```
import React from 'react';
import ReactDOM from 'react-dom/client';
function Football() {
  const shoot = (a) => {
    alert(a);
  }
  return (
    <button onClick={() => shoot("Goal!")}>Take the shot!</button>
  );
}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Football />);
```

# React Event Object

- Event handlers have access to the React event that triggered the function.
- Arrow Function: Sending the event object manually:

```
import React from 'react';
import ReactDOM from 'react-dom/client';
function Football() {
  const shoot = (a, b) => {
    alert(b.type);
    /*
    'b' represents the React event that triggered the function,
    in this case the 'click' event
    */
  }
  return (
    <button onClick={(event) => shoot("Goal!", event)}>Take the shot!</button>
  );
}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Football />);
```

- This mechanism is useful when we need to deal with the user's event properties.

# React Hooks

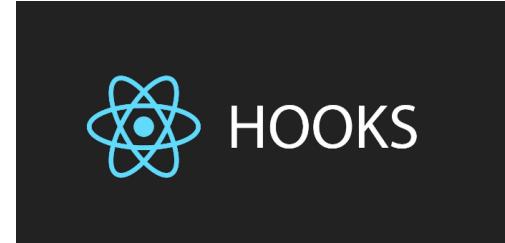
---

## ◉ React Hooks

- React v16.8에 추가된 기능
- React function component 안에서 작업함
- state와 react의 기능 활용 가능.
- props, state, context refs, 생명주기에 대해 보다 직관적인 API 제공
- 주의점 : 상위 Component에서 호출, 반복문이나 조건문 등에서 사용 주의.

## ◉ React Hook의 종류

- useState / useEffect / useContext
- useReducer, useCallback, useNavigate, useMemo
- Custom Hooks



# Hook 종류

---

- ◉ **useState** : 상태(state) 값을 생성, 변경 관리
- ◉ **useEffect** : 컴포넌트 렌더링 이후에 특정한 작업(data fetching, DOM 조작 등) 수행
- ◉ **useContext** : 리액트의 Context API를 사용하여, 컴포넌트 트리 안에서 전역적인 상태 공유
- ◉ **useCallback** : 함수를 메모이제이션(Memoization)하여 성능 최적화를 위해 사용
  - 부모 컴포넌트가 re-rendering되어도 함수를 새로 생성하지 않고 재 사용하는 경우 사용
- ◉ **useMemo** : 값을 메모이제이션(Memoization)하여 성능 최적화를 위해 사용
  - 계산 비용이 큰 연산을 수행하거나, 동일한 값을 여러번 계산하지 않도록 할 때 사용
- ◉ **useRef**
  - 컴포넌트에서 DOM 요소를 선택하거나 참조할 때 사용. ref 객체를 생성하여 DOM 요소에 접근 가능
- ◉ **useLayoutEffect** : useEffect와 비슷하게 작동. 렌더링된 후 동기적으로 실행
  - 보통 DOM 조작과 관련된 작업에 사용

# useState Hook

## ● useState Hook

- function component에서 state를 생성 및 관리를 위한 hook
- useState hook은 배열을 리턴 (Destructuring)
  - `const [ state변수, set함수 ] = useState( state변수_초기값 )`
- 중첩 component 사이에서는 가장 상위 요소에서 state 관리, 하위에는 props를 이용하여 전달

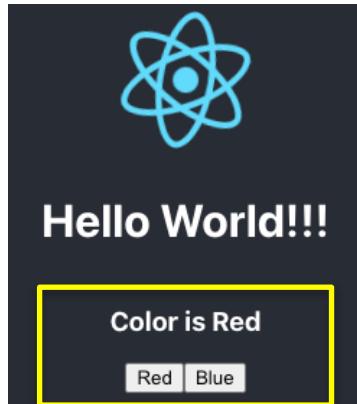
```
import {useState} from 'react';
...
const [color, setColor] = useState('Red');
...
<button type='button' onClick={() => setColor('Blue')}>Blue</button>
```

## ● 사용방법

- import useState
- initialize : 새로운 state 변수 초기화
- return values : state 변수명 , state 수정 함수 리턴

# LAB : useState Hook

- ◎ 새로운 Component 생성
  - color state 생성
- ◎ App.js에 component 추가



```
import {useState} from 'react';  
  
export default function Color(){  
  const [color, setColor] = useState('Red');  
  const changeColor = (c) => { setColor(c) };  
  return (  
    <div class='colorgroup'>  
      <h3>Color is {color}</h3>  
      <button type='button' onClick={()=>  
        changeColor('Red')}>Red</button>  
      <button type='button' onClick={()=>  
        setColor('Blue')}>Blue</button>  
    </div>  
  );  
};  
...  
import Color from './Color.js';  
  
function App() {  
  return (  
    ...  
    <Hello />  
    <Color />  
    ...  
  );  
}  
App
```

# useState Hook 사용 예

- ◉ 여러 State 사용
- ◉ 객체로 저장 가능
- ◉ state를 업데이트할 때는 반드시 변경함수(setState)를 사용해야 함.
  - color = 'Blue' // 사용할 수 없음!

```
const [brand, setBrand] = useState('HYUNDAI');
const [model, setModel] = useState('GENESIS');
const [year, setYear] = useState(2008);
const [color, setColor] = useState('Black');
```

```
const [car, setCar] = useState({
  brand: "HYUNDAI",
  model: "GENESIS",
  year: "2008",
  color: "Black"
});

const updateColor = () => {
  setCar(previousState => {
    return { ...previousState, color: "Blue" }
  });
}
```

# useState 실습(1/3)

- ◉ React Bootstrap 공식문서 참고

- <https://react-bootstrap.netlify.app/>



- ◉ 설치

```
npm install react-bootstrap bootstrap
```

- ◉ CSS 추가

```
import 'bootstrap/dist/css/bootstrap.min.css';
```

- ◉ import

```
import Button from 'react-bootstrap/Button';  
  
// or less ideally  
import { Button } from 'react-bootstrap';
```

# useState 실습(2/3)

- 초기 데이터를 이용한 리스트 작성
- 삭제 기능 구현

Studio Courses		
1	Open Source Studio	<button>Delete</button>
2	Programming Studio	<button>Delete</button>
3	Coding Studio	<button>Delete</button>

```
import './component.css';
import Container from 'react-bootstrap/Container';
...
import { useState } from "react";

const CourseList = () => {
  const initialCourses = [
    { id: "1", name: "Open Source Studio" },
    { id: "2", name: "Programming Studio" },
    { id: "3", name: "Coding Studio" }
  ];

  const [courses, setCourses] = useState(initialCourses);
  const delfunc = (id) => {
    if (window.confirm("정말로 삭제하실래요? ")) {
      const newList = courses.filter((c) => c.id !== id);
      setCourses(newList);
    }
  }
  return (
    <Container>
      {courses.map((course) => (
        <Row>
          <Col>{course.id}</Col>
          <Col xs={6}>{course.name}</Col>
          <Col><Button variant="danger" onClick={() => delfunc(course.id)}>Delete</Button></Col>
        </Row>
      ))}
    </Container>
  );
}

export default CourseList;
```

# useState 실습(3/3)

## ◎ props를 통한 sub component 작성

```
import { useState } from "react";
import CourseItem from './CourseItem';
const CourseList = () => {
  const initialCourses = [
    { id: "1", name: "Open Source Studio" },
    { id: "2", name: "Programming Studio" },
    { id: "3", name: "Coding Studio" }
  ];

  const [courses, setCourses] = useState(initialCourses);
  const delfunc = (id) => {
    if (window.confirm("정말로 삭제하실래요? ")) {
      const newList = courses.filter((c) => c.id !== id);
      setCourses(newList);
    }
  }
  return (
    <Container>
      {courses.map((course) => <CourseItem data={course} delfunc={delfunc} />)}
    </Container>
  );
};

export default CourseList;
```

CourseList.js

```
import './component.css';
import Row from 'react-bootstrap/Row';
import Col from 'react-bootstrap/Col';
import Button from 'react-bootstrap/Button';

const CourseItem = (props) => {
  const data = props.data;
  return (
    <Row key={data.id}>
      <Col>{data.id}</Col>
      <Col xs={6}>{data.name}</Col>
      <Col><Button variant="danger" onClick={() => props.delfunc(data.id)}>Delete</Button></Col>
    </Row>
  );
}

export default CourseItem;
```

CourseItem.js

# useEffect Hook

## ◉ component 내에서 side effect를 수행할 수 있음.

- DOM이 업데이트 된 후 실행
- 데이터 가져오기, DOM 업데이트, 타이머 등
- useEffect()는 기본적으로 렌더링할 때마다 실행
- Class component의 LifeCycle 중  
componentDidMount, componentDidUpdate,  
componentWillUnmount를 하나의 API로 통합

## ◉ 사용방법

- useEffect(function, [optional dependency]) 함수 사용
- first rendering 1회만 실행하려면 빈 배열 추가 []

```
// 기본형 (매번 rendering)
useEffect(() => {
  //Runs on every render
});
```

```
// first rendering 에서만 실행
useEffect(() => {
  //Runs only on the first render
}, []);
```

# useEffect Hook

## ◎ 사용 방법

1. 매 렌더링마다 실행
2. 첫 번째 렌더링 때 1회 실행
3. 첫 번째 렌더링에 실행되고,  
특정값 업데이트될 때 실행

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

function MyTimer(){

    // Case 1 매 렌더링마다 실행
    useEffect(()=> {
        // side effect
    });

    // Case 2 first rendering에서만 실행
    useEffect(()=> {
        // side effect
    }, []);

    // Case 3 first & 특정값 변경시 실행
    useEffect(()=> {
        // side effect
    }, [value]);
}
```

# LAB : useEffect Hook(1)

- Timer를 이용한 useEffect 구현
  - count state 생성
  - setTimeout()을 사용한 Timer구현



```
import { useState, useEffect } from "react";
export default function Timer() {
  console.log("Timer rendered!");
  const [count, setCount] = useState(0);

  useEffect(() => {
    setTimeout(() => {
      setCount((count) => count + 1);
    }, 1000);
  });

  return <h1> {count} times!</h1>;
}

useEffect(() => {
  setTimeout(() => {
    setCount((count) => count + 1);
  }, 1000);
}, [ ]);
```

Timer.js

매번 렌더링에서 first 렌더링으로 변경

# LAB : useEffect Hook(2)

## ◉ timer를 이용한 useEffect 구현

- timer의 메모리 누수 등의 정리가 필요할 때 return 함수 추가
- 소스 실행 시 timer 멈춤(first rendering만 실행)
- Timer기능구현 - count가 변할 때만  
useEffect()가 실행하도록 하려면?
  - useEffect의 dependency추가



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

export default function Timer() {

  console.log("Timer rendered!");
  const [count, setCount] = useState(0);

  useEffect(() => {
    let timer = setTimeout(() => {
      setCount((count) => count + 1);
    }, 1000);

    return () => clearTimeout(timer)
  }, []);

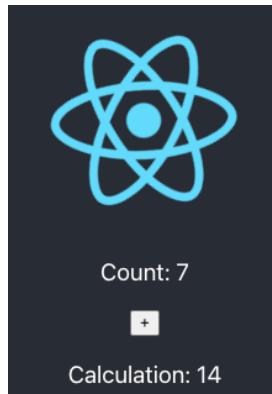
  return <h1> {count} times!</h1>;
}
```

Timer.js

# LAB : useEffect Hook(3)

## ◉ 이벤트를 통한 useEffect

- count, calculation state 생성
- onclick을 통한 useEffect



```
import { useState, useEffect } from "react";           CountEffect.js

export default function CountEffect() {

  console.log ("CountEffect rendered !");
  const [count, setCount] = useState(0);
  const [calculation, setCalculation] = useState(0);

  useEffect(() => {
    console.log ("Call setCalculation !");
    setCalculation(() => count * 2);
  }, [count]); // <- add the count variable here

  return (
    <>
      <p>Count: {count}</p>
      <button onClick={() => setCount((c) => c + 1)}>+</button>

      <p>Calculation: {calculation}</p>
    </>
  );
}
```

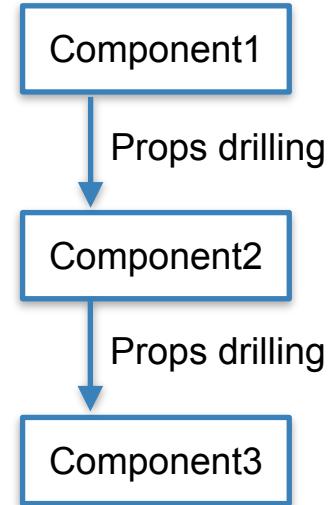
# useContext Hook

## ◉ useContext Hook

- state를 전역적으로 관리하는 hook
- useState hook와 함께 사용할 때 중첩된 component 사이에 상태 공유가 가능함
- **props drilling 대신 사용**
  - useState hook로만 중첩된 component 사이에서 공유할 때는 props로 전달

## ◉ 사용방법

- Import : import { useState, createContext } from "react";
- Create context : const UserContext = **createContext()**
- Context Provider
- Use the useContext Hook in a child component



# useContext Hook 예제

## ◎ Props drilling vs useContext Hook

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

export default function PropDrilling(){
    const [name, setName] = useState("Jerry");
    return (
        <>
            <h1>Hello {name}!</h1>
            <PropsDrillingCom1 name={name} />
        </>
    );
}

function PropsDrillingCom1({name}){
    return (
        <>
            <h1>Component1 : Hello {name}</h1>
        </>
    );
}
```

```
//using useContext hook
import { useState, useContext, createContext } from "react";

const UserContext = createContext();

export default function PropNoDrilling() {
    const [name, setName] = useState("Jerry");
    return (
        <UserContext.Provider value={name}>
            <h1>Hello {name}!</h1>
            <PropNoDrillingCom1 />
        </UserContext.Provider>
    );
}

function PropNoDrillingCom1() {
    const name = useContext(UserContext);
    return (
        <>
            <h1> Component1(UseContext) : Hello {name}</h1>
        </>
    );
}
```

Hello Jerry!

Component1(UseContext) : Hello Jerry

# useContext Hook 예제2(1/4)

- Context와 Provider를 분리하여 필요한 곳에서 사용하기

- HeaderContext.js 생성
- HeaderProvider.js 생성
- Content.js 생성 (useContext 사용)
- App.js 수정

HeaderContext.js

```
import {createContext} from "react";
const HeaderContext = createContext({color : "red"});
export default HeaderContext;
```

# useContext Hook 예제2(2/4)

## ◎ Context와 Provider를 분리하여 필요한 곳에서 사용하기

- HeaderContext.js 생성
- **HeaderProvider.js 생성**
- Content.js 생성 (useContext 사용)
- App.js 수정

```
import {useState} from "react";
import HeaderContext from "./HeaderContext";

const HeaderProvider = ({children}) => {

    const changeColor = (c) => {
        setColorState((prevState) => {
            return {
                ...prevState,
                color : c
            }
        });
        alert(c + " clicked!");
        //console.log(c);
    };

    const initialColorState = {
        color : "Red",
        changeColor
    };

    const [colorState, setColorState] = useState(initialColorState);

    return (
        <HeaderContext.Provider value={colorState}>{children}</HeaderContext.Provider>
    );
}

export default HeaderProvider;
```

HeaderProvider.js

# useContext Hook 예제2(3/4)

## ● Context와 Provider 를 분리하여 필요한 곳에 서 사용하기

- HeaderContext.js 생성
- HeaderProvider.js 생성
- Content.js 생성  
(useContext 사용)
- App.js 수정

Content.js

```
import { useContext } from "react";
import HeaderContext from "./HeaderContext";

export default function HeaderComponent() {
  const { changeColor, color } = useContext(HeaderContext);

  // view 값 조회
  console.log(color);

  return (
    <div>
      <button onClick={() => changeColor("Yellow")}>Yellow</button>
      <button onClick={() => changeColor("Red")}>Red</button>
      <button onClick={() => changeColor("Blue")}>Blue</button>
    </div>
  );
}
```

# useContext Hook 예제2(4/4)

## ● Context와 Provider를 분리하여 필요한 곳에서 사용하기

- HeaderContext.js 생성
- HeaderProvider.js 생성
- Content.js 생성 (useContext 사용)
- [App.js 수정](#)

App.js

```
...
import HeaderProvider from './Context/HeaderProvider';
import Content from './Context/Content';

function App() {
  return (
    <div className="App">
      <h1>
        Studio Courses
      </h1>
      <HeaderProvider>
        <Content />
      </HeaderProvider>
    </div>
  );
}

export default App;
```

# useRef Hook

## ● 특징

- 렌더링에 필요하지 않은 값을 참조할 수 있는 hook
- Ref로 값 참조
- Ref로 DOM 조작
- Ref로 콘텐츠 재생성 피할 수 있음
- [단점] Custom component에 대한 ref는 얻을 수 없음

`const` 변수명 = useRef(`초기값`)

## ● useRef hook

- current라는 key가 있는 객체를 반환 { `current : 초기값` }
- `ref.current` 값이 변경되어도 React는 Component를 다시 렌더링 하지 않음.
- 초기화를 제외한 렌더링 중에 `ref.current`를 사용하지 않는 것이 좋음

# useRef Hook 예제1

## ◉ ref로 값 참조

- useRef와 useEffect 사용
- render에 종속된 변수가 아니면, 값이 변경되어도 Component rendering은 일어나지 않음
- 렌더링 중에 ref.current를 사용하지 않는 것이 좋음

```
import { useRef } from 'react';

export default function RefTest() {
  let ref = useRef(0);

  function handleClick() {
    ref.current = ref.current + 1;
    alert('You clicked ' + ref.current + ' times!');
  }

  return (
    <button onClick={handleClick}>
      Click me!
    </button>
  );
}
```

# useRef Hook 예제2

- Rendering에 종속된 Ref 사용시
  - 불가피하게 re-rendering이 일어남
  - re-rendering 예측 불가이므로 가능한 사용하지 않도록 함!

```
import { useState, useRef, useEffect } from "react";
function RefTest() {
  const [inputValue, setInputValue] = useState("");
  const count = useRef(0);

  useEffect(() => {
    count.current = count.current + 1;
  });

  return (
    <>
      <input
        type="text"
        value={inputValue}
        onChange={(e) => setInputValue(e.target.value)}
      />
      <h1>Render Count: {count.current}</h1>
    </>
  );
}

export default RefTest;
```

# useRef Hook 예제3

## ◎ DOM element 참조

- 특정 DOM 요소에 ref 설정

```
import { useRef } from "react";
import ReactDOM from "react-dom/client";

function App() {
  const inputElement = useRef();

  const focusInput = () => {
    inputElement.current.focus();
  };

  return (
    <>
      <input type="text" ref={inputElement} />
      <button onClick={focusInput}>Focus Input</button>
    </>
  );
}

const root =
ReactDOM.createRoot(document.getElementById('root'));
root.render(<App />);
```

# useMemo Hook

---

## ◎ Memoization?

- value를 캐싱하여 사용하므로, 변경 사항이 없다면 새로 계산할 필요가 없음.

## ◎ useMemo hook

- 메모된 값을 리턴하는 hook
- Parent Component가 렌더링 될 때 불필요하게 해당 컴포넌트가 렌더링 되지 않도록 함.
- 현재 함수의 종속성에 변화가 있을 때 다시 수행됨.

## ◎ useMemo hook 또는 React.memo를 사용할 수 있음.

## ◎ useMemo와 useRef의 공통점 : 캐싱 활용

## ◎ useMemo는 계산 비용을 줄이기 위해, useRef는 값의 변화를 유지하기 위해 사용

```
const function_name = useMemo(() =>  
  compute_func((a, b), [a, b]));
```

# useMemo Hook 예제

```
import { useState, useMemo } from "react";
import ReactDOM from "react-dom/client";

const App = () => {
  const [count, setCount] = useState(0);
  const [todos, setTodos] = useState([]);
  const calculation = useMemo(() => expensiveCalculation(count), [count]);

  const increment = () => {
    setCount((c) => c + 1);
  };
  const addTodo = () => {
    setTodos((t) => [...t, "New Todo"]);
  };

  return (
    <div>
      <div>
        <h2>My Todos</h2>
        {todos.map((todo, index) => {
          return <p key={index}>{todo}</p>;
        })}
        <button onClick={addTodo}>Add Todo</button>
      </div>
      <hr />
      <div>
        Count: {count} <button onClick={increment}>+</button>
        <h2>Expensive Calculation</h2>
        {calculation}
      </div>
    </div>
  );
};
```

```
const expensiveCalculation = (num) => {
  console.log("Calculating...");
  for (let i = 0; i < 1000000000; i++) {
    num += 1;
  }
  return num;
};

const root =
ReactDOM.createRoot(document.getElementById('root'));
root.render(<App />);
```

# useCallback Hook

## ◎ 특징

- 메모화된(memoized) callback 함수 리턴
  - Memoization : 값을 캐싱하여 다시 계산할 필요가 없음
- 종속성 있는 것 중 업데이트될 때 실행되어 성능이 향상됨
- 속성이 변경되지 않는다면 렌더링 되지 않음

## ◎ useMemo은 메모화된 값을, useCallback은 메모화된 함수 리턴

```
const function_name = useCallback(() => {  
    ...  
}, [initial_value]);
```

# useCallback Hook 예제

```
import { memo } from "react";           Todos.js

const Todos = ({ todos, addTodo }) => {
  console.log("todos render");
  return (
    <>
      <h2>My Todos</h2>
      {todos.map((todo, index) => {
        return <p key={index}>{todo}</p>;
      })}
      <button onClick={addTodo}>Add Todo</button>
    </>
  );
};

export default memo(Todos);
```

```
import { useState, useCallback } from "react";
import Todos from './Todos';

const CallbackParent = () => {
  console.log('CallbackParent render!');
  const [count, setCount] = useState(0);
  const [todos, setTodos] = useState([]);

  const increment = () => {
    setCount((c) => c + 1);
  };

  const addTodo = useCallback(() => {
    setTodos((t) => [...t, "New Todo"]);
  }, [todos]);

  return (
    <div>
      <Todos todos={todos} addTodo={addTodo} />
      <hr />
      <div>
        Count: {count}
        <button onClick={increment}>+</button>
      </div>
    </div>
  );
};

export default CallbackParent;
```

# useReducer Hook

---

- ◉ useState hook과 비슷함.
- ◉ custom state logic 가능
  - 복잡한 로직 + 다수 state 을 다룰 수 있음.

## ◉ 사용법

- useReducer(reducer, 상태초기값들)
- [state, dispatch method]를 리턴
  - state : 현재 state 값
  - dispatch : state를 업데이트하고 리렌더링을 촉발하는 함수

```
const [todos, dispatch] = useReducer(reducer, initialTodos);
```

# useReducer Hook 예제

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

const initialTodos = [
  {
    id: 1,
    title: "Todo 1",
    complete: false,
  },
  {
    id: 2,
    title: "Todo 2",
    complete: false,
  },
];

const reducer = (state, action) => {
  switch (action.type) {
    case "COMPLETE":
      return state.map((todo) => {
        if (todo.id === action.id) {
          return { ...todo, complete: !todo.complete };
        } else {
          return todo;
        }
      });
    default:
      return state;
  }
};
```

```
function Todos() {
  const [todos, dispatch] = useReducer(reducer, initialTodos);

  const handleComplete = (todo) => {
    dispatch({ type: "COMPLETE", id: todo.id });
  };

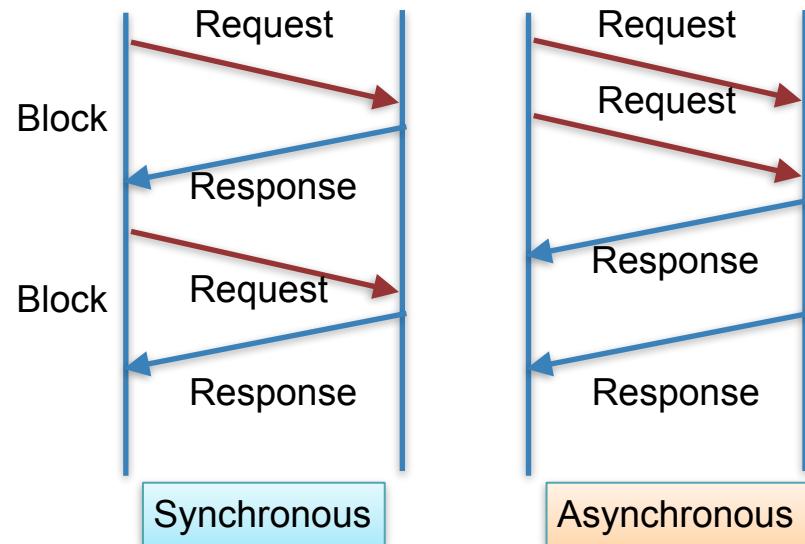
  return (
    <>
      {todos.map((todo) => (
        <div key={todo.id}>
          <label>
            <input
              type="checkbox"
              checked={todo.complete}
              onChange={() => handleComplete(todo)}
            />
            {todo.title}
          </label>
        </div>
      ))}
    </>
  );
}
```

19

# React Library

# AXIOS

- Promise 기반 HTTP 클라이언트 라이브러리(비동기 통신)
- Axios는 요청과 응답 모두 JSON형식으로 자동 변환
- JS : AJAX(Javascript And XML) library, Fetch API(Built-in library)
- 설치하기  
**% npm install axios**



# AXIOS 사용법

## ◎ HTTP통신 Method (GET/POST/PUT/DELETE)

- axios.get(url, [config])
- axios.post(url, data, [config])
- .then() 성공시 실행
- .catch() 실패시 에러 처리

```
const axios_get = () => {
  axios.get("http://localhost:8080/get")
    .then((response) => {
      console.log(response);
    })
    .catch((error) => {
      console.log(error);
    })
}
```

```
const axios_post = () => {
  const data = {
    name: 'name',
    age: 23
  }
  axios.post("http://localhost:8080/post", data)
    .then((response) => {
      console.log(response);
    })
    .catch((error) => {
      console.log(error);
    })
}
```

# AXIOS 사용법

## ◎ HTTP통신 Method (GET/POST/PUT/DELETE)

- axios.put(url [,data] [,config])
- axios.delete(url[, config])
- .then() 성공시 실행
- .catch() 실패시 에러 처리

```
const axios_put = () => {
  const data = {
    age: 25
  }
  axios.put("http://localhost:8080/put", data)
    .then((response) => {
      console.log(response);
    })
    .catch((error) => {
      console.log(error);
    })
}
```

```
const axios_delete = () => {
  axios.delete("http://localhost:8080/delete")
    .then((response) => {
      console.log(response);
    })
    .catch((error) => {
      console.log(error);
    })
}
```

# 순서가 있는 비동기 통신

## ● 비동기통신을 단계별로 처리할 경우

- 비동기 통신은 순서를 보장하지 않음
- 요청1이 성공한 후  
비동기 통신2가 실행 되어야 하는 경우

## ● 해결방법

- 비동기 통신 후 성공인 경우 다시 요청
- async(), await() 사용

<https://axios-http.com/>

해결방안 1

```
// 2개의 api를 가져와야 하는 예제
import axios from "axios";

const GetDataAxios = () => {
  axios.get('https://66ff38132b9aac9c997e8ebd.mockapi.io/api/oss/users/1')
    .then((response) => {
      const data = response.data;
      const userId = data.id;
      console.log(data);
      axios.get('https://66ff38132b9aac9c997e8ebd.mockapi.io/api/oss/user_courses/?user_id=' + userId)
        .then((response) => {
          console.log("Response >>", response.data)
        })
        .catch(() => {
        })
    })
    .catch((error) => {
      console.log("Error >>", error);
    })
}

export default GetDataAxios;
```

# AXIOS 사용법

## ◎ async(), await() 사용

- 동기적인 호출방식 보장
- 코드가 간결함

```
import axios from "axios";  
  
const AxiosAsync = () => {  
    const handleClick = async () => {  
        let result = await axios.get('https://  
66ff38132b9aac9c997e8ebd.mockapi.io/api/oss/users/1')  
        const data = result.data;  
        const userId = data.id;  
        console.log(data);  
  
        let result2 = await axios.get('https://  
66ff38132b9aac9c997e8ebd.mockapi.io/api/oss/user_courses/?  
user_id=' + userId)  
        console.log(result2.data);  
    }  
  
    return (  
        <>  
            <h3>데이터 가져오기(AXIOS)</h3>  
            <button onClick={handleClick}>데이터가져오기</  
button>  
        </>  
    )  
}  
  
export default AxiosAsync;
```

해결방안 2

20

# React Forms

# Handling Forms(1/4)

## ◉ React는 Component로 처리

- state 변수로 각 요소 값 처리(useState hook)
- onChange() 함수로 값 제어

## ◉ Validation Check – onSubmit

## ◉ 다수 Text field 처리 방법

## ◉ HTML과 다른 요소

- textarea, select

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

export default function Form1() {
  const [name, setName] = useState("");
  return (
    <form>
      <label>Enter your name:</label>
      <input
        type="text"
        value={name}
        onChange={(e) => setName(e.target.value)}
      />
    </form>
  )
}
```

# Handling Forms(2/4)

## ● Validation Check – onSubmit

Studio Courses

Enter your name:  Save

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

export default function Form1() {
  const [name, setName] = useState("");
  const handleSubmit = (event) => {
    event.preventDefault();
    alert(`The name you entered was: ${name}`);
  }
  return (
    <form onSubmit={handleSubmit}>
      <label>Enter your name:</label>
      <input
        type="text"
        value={name}
        onChange={(e) => setName(e.target.value)}
      />
      </label>
      <button type="submit">Save</button>
    </form>
  )
}
```

# Handling Forms(3/4)

## ◎ 다수 Text field 처리 방법

- event.target.name 와 event.target.value를 사용

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

export default function Form3() {
  const [inputs, setInputs] = useState({});

  const handleChange = (event) => {
    const name = event.target.name;
    const value = event.target.value;
    setInputs(values => ({ ...values, [name]: value }))
  }

  const handleSubmit = (event) => {
    event.preventDefault();
    alert("Name: " + inputs.username +
      ", Age: " + inputs.age);
  }
}
```

## Studio Courses

Enter your name:   
Enter your age:  제출

```
return (
  <form onSubmit={handleSubmit}>
    <label>Enter your name:
      <input
        type="text"
        name="username"
        value={inputs.username || ""}
        onChange={handleChange}
      />
    </label><br />
    <label>Enter your age:
      <input
        type="number"
        name="age"
        value={inputs.age || ""}
        onChange={handleChange}
      />
    </label>
    <input type="submit" />
  </form>
);
```

# Handling Forms(4/4)

- ◎ HTML 태그와 약간 다른 요소
  - <textarea>, <select>

```
//in React
<textarea value={textarea} onChange={handleChange} />

//in HTML
<textarea> value </textarea>
```

```
//in React
<select value="Ford" onChange={handleChange}>
  <option value="Ford">Ford</option>
  <option value="Volvo">Volvo</option>
  <option value="Fiat">Fiat</option>
</select>
```

```
//in HTML
<select onChange={handleChange}>
  <option value="Ford" selected>Ford</option>
  <option value="Volvo">Volvo</option>
  <option value="Fiat">Fiat</option>
</select>
```

# React Form Library

- Form을 위한 다양한 라이브러리로 효율적인 개발
- Form validation
  - Validex
- Form UI Libaray
  - MUI, Ant Design, Chakra
- Form library
  - React Hook Form
  - Formik



# Adding Forms in React

- Just like in HTML, React uses forms to allow users to interact with the web page.
  - Add a form that allows users to enter their name:

```
import React from 'react';
import ReactDOM from 'react-dom/client';

function MyForm() {
  return (
    <form>
      <label>Enter your name:
        <input type="text" />
      </label>
    </form>
  )
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<MyForm />);
```

# Handling Forms

- Handling forms is about how you handle the data when it changes value or gets submitted.
- In HTML, form data is usually handled by the DOM.
- In React, form data is usually handled by the components.
- When the data is handled by the components, all the data is stored in the component state.
- You can control changes by adding event handlers in the onChange attribute.
- We can use the useState Hook to keep track of each inputs value and provide a "single source of truth" for the entire application.

# Handling Forms (2)

## ◉ Example

- Use the useState Hook to manage the input:

```
import { useState } from 'react';
import ReactDOM from 'react-dom/client';
function MyForm() {
  const [name, setName] = useState("");
  return (
    <form>
      <label>Enter your name:
        <input
          type="text"
          value={name}
          onChange={(e) => setName(e.target.value)}
        />
      </label>
    </form>
  )
}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<MyForm />);
```

# Submitting Forms

- You can control the submit action by adding an event handler in the `onSubmit` attribute for the `<form>`:
- Add a submit button and an event handler in the `onSubmit` attribute:

```
function MyForm() {
  const [name, setName] = useState("");
  const handleSubmit = (event) => {
    event.preventDefault();
    alert(`The name you entered was: ${name}`);
  }
  return (
    <form onSubmit={handleSubmit}>
      <label>Enter your name:
        <input
          type="text"
          value={name}
          onChange={(e) => setName(e.target.value)}
        />
      </label>
      <input type="submit" />
    </form>
  )
}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<MyForm />);
```

# Multiple Input Fields

- You can control the values of more than one input field by adding a name attribute to each element.
- To access the fields in the event handler use the event.target.name and event.target.value syntax.
- To update the state, use square brackets [bracket notation] around the property name.
  - Write a form with two input fields:

```
...
function MyForm() {
  const [inputs, setInputs] = useState({});

  const handleChange = (event) => {
    const name = event.target.name;
    const value = event.target.value;
    setInputs(values => ({...values, [name]: value}))
  }

  const handleSubmit = (event) => {
    event.preventDefault();
    alert("Name: " + inputs.username + ", Age: " + inputs.age));
  } // continue in the next slide
}
```

# Multiple Input Fields

```
...
return (
  <form onSubmit={handleSubmit}>
    <label>Enter your name:
      <input
        type="text"
        name="username"
        value={inputs.username || ""}
        onChange={handleChange}
      />
    </label>
    <label>Enter your age:
      <input
        type="number"
        name="age"
        value={inputs.age || ""}
        onChange={handleChange}
      />
    </label>
    <input type="submit" />
  </form>
)
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<MyForm />);
```

# Textarea

- The textarea element in React is slightly different from ordinary HTML.
- In HTML the value of a textarea was the text between the start tag `<textarea>` and the end tag `</textarea>`.
- In React the value of a textarea is placed in a `value` attribute. We'll use the `useState` Hook to manage the value of the textarea:
  - A simple textarea with some content:

```
function MyForm() {
  const [textarea, setTextarea] = useState(
    "The content of a textarea goes in the value attribute"
);
  const handleChange = (event) => {
    setTextarea(event.target.value)
}
  return (
    <form>
      <textarea value={textarea} onChange={handleChange}> />
    </form>
  )
}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<MyForm />);
```

# Select

- A drop down list, or a select box, in React is also a bit different from HTML.
- in HTML, the selected value in the drop down list was defined with the selected attribute:
- In React, the selected value is defined with a value attribute on the select tag:
  - A simple select box, where the selected value "Volvo" is initialized in the constructor:

```
...
function MyForm() {
  const [myCar, setMyCar] = useState("Volvo");
  const handleChange = (event) => {
    setMyCar(event.target.value)
  }
  return (
    <form>
      <select value={myCar} onChange={handleChange}>
        <option value="Ford">Ford</option>
        <option value="Volvo">Volvo</option>
        <option value="Fiat">Fiat</option>
      </select>
    </form>
  )
}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<MyForm />);
```

21

# React Router

# React Router

---

- ◉ CRA(create-react-app)은 Page Routing을 포함하지 않음
- ◉ React Router(`react-router-dom`) library 사용
- ◉ React Router 설치
  - Application root 디렉터리에서 실행  
% `npm i react-router-dom`



# React Router 설치

- 새로운 앱 생성 및 react-router-dom 설치
- App.js에 import statement : BrowserRouter, Routes, Route

```
import { BrowserRouter as Router, Route, Routes } from 'react-router-dom';
```

## ● Router 설정 실습

- src/pages/ 폴더 하위 Component 생성
  - Home.js
  - About.js

```
<BrowserRouter>
  <Routes>
    <Route path="/" element={<Home />} />
    <Route path="/about" element={<About />} />
  </Routes>
</BrowserRouter>
```

# React Router 실습

## ◎ App.js

```
import './App.css';
import { BrowserRouter as Router, Route, Routes } from 'react-router-dom';
import Home from './pages/Home';
import About from './pages/About';

function App() {
  return (
    <Router>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
      </Routes>
    </Router>
  );
}

export default App;
```

# React Router 실습

- ◉ Home.js
- ◉ Link component : HTML Anchor tag 역할

```
import {Link} from 'react-router-dom';

export default function Home() {
  return (
    <div style={{ padding: 20 }}>
      <h2>Home View</h2>
      <p>Lorem ipsum dolor sit amet, consectetur adip.</p>
      <p><Link to='/about'>go to About page</Link></p>
    </div>
  );
}
```

# React Router 실습

- ◉ About.js
- ◉ Link component : HTML Anchor tag 역할

```
import {Link} from 'react-router-dom';

export default function About() {
  return (
    <div style={{ padding: 20 }}>
      <h2>About View</h2>
      <p>Lorem ipsum dolor sit amet, consectetur adip.</p>
      <p><Link to='/' >go to Home page</Link></p>
    </div>
  );
}
```

# React Router 실습

- ◎ 그 외 모든 페이지 처리
- ◎ App.js에서 작업
  - component 추가

App.js

```
...
function NoMatch() {
  return (
    <div style={{ padding: 20 }}>
      <h2>404: Page Not Found</h2>
      <p>Lorem ipsum dolor sit amet, consectetur adip.</p>
    </div>
  );
}

function App() {
  return (
    <Router>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
        <Route path "*" element={<NoMatch />} />
      </Routes>
    </Router>
  );
...
}
```

# Outlet

- 중첩 라우팅을 통해 상위 컴포넌트의 레이아웃
- 다른 형태의 페이지를 제작하고 싶으면?

```
...
function App() {
  return (
    <Router>
      <Routes>
        <Header />
        <Sidebar />
        <Route path="/" element={<Home />} />
        <Route path="/a" element={<PageA />} />
        <Route path="/b" element={<PageB />} />
        <Route path="/c" element={<PageC />} />
      </Routes>
    </Router>
  );
}
...
```

# Outlet 실습

## ◎ 중첩 상위 레이아웃 제작하여 Routing에 의해 변경되는 부분에 Outlet 사용

```
import { Outlet } from 'react-router-dom';
import React from 'react'

export default function HomeOutlet() {
  return (
    <>
      <div style={{backgroundColor: "yellow" }}>Header</div>
      <div style={{backgroundColor: "magenta" }}>SideBar</div>
      <div id="contents" style={{backgroundColor: "PapayaWhip" }}>

        <Outlet />

      </div>
    </>
  )
}
```

HomeOutlet.js

```
function App() {
  return (
    <Router>
      <Routes>
        <Route path="/" element={<HomeOutlet />}>
          <Route path="/" element={<Home />} />
          <Route path="/about" element={<About />} />
          <Route path="*" element={<NoMatch />} />
        </Route>
        <Route path="/blog" element={<Blog />} />
      </Routes>
    </Router>
  );
}

...
```

App.js

# Outlet 실습

- ◎ /blog만 디자인이 다른 이유는?

```
function App() {
  return (
    <Router>
      <Routes>
        <Route path="/" element={<HomeOutlet />}>
          <Route path="/" element={<Home />} />
          <Route path="/about" element={<About />} />
          <Route path="*" element={<NoMatch />} />
        </Route>
        <Route path="/blog" element={<Blog />} />
      </Routes>
    </Router>
  );
}
```

App.js

