React Native

React Native로 Native App 만들기

# Development Environment Settings

## 맥(macOS) 개발 환경

### 홈브루(Homebrew) 설치

* 홈브루란? 맥용 패키지 관리자
* 다운로드
* url: <https://brew.sh/>
* Terminal에서 아래 명령 실행

/usr/bin/ruby -e “$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)”

### 노드 설치

* Brew install node@12
* Node --version

### 왓치맨 설치

* 왓치맨은 특정 디렉토리나 파일을 감시하다가, 변경이 발생하면, 특정 동작을 실행(Trigger)하도록 하는 역할을 수행. 리액트 네이티브는 소스 변경이 발생하면 자동적으로 빌드하고 디바이스 혹은 시뮬레이터에 업로드하기 위해 왓치맨 사용.
* 설치
* Brew install watchman
* watchman -version

### 리액트 네이티브 CLI 설치

* npm install -g react-native-cli

### XCode 설치

* iOS 모바일 앱을 개발하기 위해서는 맥OS와 iOS 개발 툴인 XCode가 반드시 필요하다.
* 앱스토어에서 설치

### 코코아포드 설치

* 코코아포드(Cocoapods)는 iOS 개발에 사용되는 의존성 관리자.
* Objective-C나 Swift로 iOS를 개발할 때, 사용되는 라이브러리를 설치하거나 관리할 때 사용
* 설치
* Sudo gem install cocoapods

### JDK 설치

* 안드로이드를 개발하기 위해서 안드로이드 개발 언어인 자바가 필요함. 자바로 개발하기 위해서 JDK가 필요
* 설치
* Brew tap AdoptOpenJDK/openjdk
* Brew cast install adoptopenjdk8
* Java -version

### 안드로이드 스튜디오 설치

* <https://developer.android.com/studio/index.html>
* 환경 설정
* .bash\_profile 파일 열고 아래 내용 추가

export ANDROID\_HOME=안드로이드SDK위치/Android/sdk

export PATH=$PATH:$ANDROID\_HOME/emulator

export PATH=$PATH:$ANDROID\_HOME/tools

export PATH=$PATH:$ANDROID\_HOME/tools/bin

export PATH=$PATH:$ANDROID\_HOME/platform-tools

* macOS Catalina(version 10.5)부터 zsh이 기본 터미널 쉘로 지정되었음. ~/.zshrc를 열고 수정.

## 윈도우(Windows) 개발 환경

### 초콜렛티(Chocolatey) 설치

* 윈도우 패키지 매니저(Package Manager for Windows users)
* 패키지를 관리하는 작업을 자동화, 안전처리 하기 위해 사용되는 도구
* 다운로드
* URL: <https://chocolatey.org/install>
* Windows XP | News Wikia | Fandom> PowerShell 검색 > 관리자 권한으로 실행 > 아래 명령 실행

Set-ExecutionPolicy Bypass -Scope Process -Force; [System.Net.ServicePointManager]::SecurityProtocol = [System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-Object System.Net.WebClient).DownloadString('https://chocolatey.org/install.ps1'))

### 노드 설치

* choco install nodejs-lts

### 파이썬 설치

* choco install -y python2

### 자바 개발 킷(jdk) 설치

* choco install -y jdk8
* java -version

### 리액트 네이티브 CLI 설치

* npm install -g react-native-cli
* react-native version

### 안드로이드 스튜디오 설치

* <https://developer.android.com/studio/index.html>
* 환경 변수 설정
* 시스템 변수 생성 ANDROID\_HOME=C:\Users\khj\AppData\Local\Android\Sdk\
* Path에 추가 🡪 %ANDROID\_HOME%\platform-tools

# React Native Project

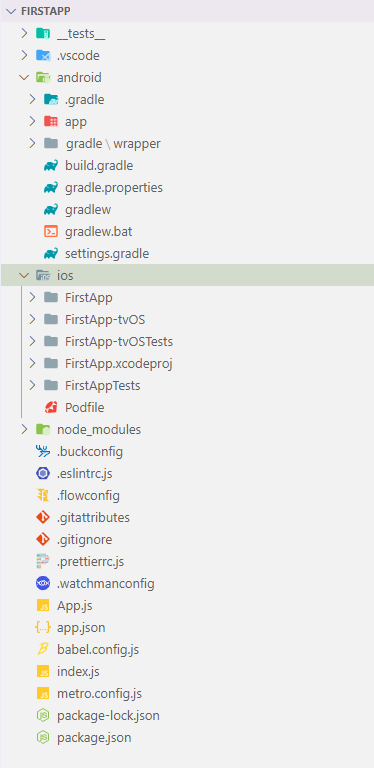
## 프로젝트 시작하기

### 프로젝트 생성

* 버전 고정: npm config set save-exact=true
* 프로젝트 생성: react-native init {ProjectName}
* 프로젝트 실행: npm run android/ios

### 폴더 및 파일

* 이미지



* React-native
* ./index.js: 리액트 네이티브 프로젝트의 시작파일. 이 파일을 시작으로 리액트 네이티브의 자바스크립트 코드가 번들링(bundling)됨
* Android
* ./android: 안드로이드 프로젝트가 담겨있는 폴더
* ./android/app/build.gradle: 안드로이드 앱을 빌드, 배포할 때 사용하는 파일
* ./android/app/src/main/java/com/firstapp/MainActivity.java, MainApplication.java: 안드로이드 앱의 메인 파일
* ./android/app/src.res: 안드로이드 앱의 아이콘 또는 시작 화면(Launch Screen) 등의 리소스를 관리하는 폴더
* iOS
* ./ios: iOS 프로젝트가 담겨 있는 폴더
* ./ios/FirstApp/AppDelegate.h, AppDelegate.m: iOS 앱의 메인 파일들
* ./ios/FirstApp/Info.plist: iOS 프로젝트의 설정 파일
* ./ios/FirstApp, scworkspace: iOS의 프로젝트를 XCode로 시작하기 위한 파일
* /ios/FirstApp/Podfile: iOS에서는 코코아 포드라는 의존성 관리자를 사용하여 라이브러리 관리, npm과 같은 역할(npm에서는 package.json을 사용하여 의존성 관리하듯이 코코아 포드에서는 Podfile을 사용하여 의존성을 관리)

### 리액트 네이티브의 컴포넌트

* 특징
* HTML 태그 대신 리액트 네이티브에서 정한 특별한 태그(컴포넌트)만 사용
* 함수형 컴포넌트(Functional Component) 기본으로 사용
* 리액트 네이티브 컴포넌트를 반환(Return)하는 함수로 구성. 이 때 반환하는 컴포넌트는 하나의 노드로 구성
* Index.js

import {AppRegistry} from 'react-native';

import App from './App';

import {name as appName} from './app.json';

AppRegistry.registerComponent(appName, () => App);

* AppRegistry.registerComponent 를 사용하여 네이티브 브릿지에서 사용할 모듈 지정
* registerComponent 함수의 첫번째 매개변수는 모듈이름 지정, 두번째 매개변수에는 처음으로 렌더링될 컴포넌트 지정.
* App.js

const App: () => React$Node = () => {

  return (

    <>

      <StatusBar barStyle="dark-content" />

      <SafeAreaView>

        <ScrollView

          contentInsetAdjustmentBehavior="automatic"

          style={styles.scrollView}>

          <Header />

          {global.HermesInternal == null ? null : (

            <View style={styles.engine}>

              <Text style={styles.footer}>Engine: Hermes</Text>

            </View>

          )}

          <View style={styles.body}>

            <View style={styles.sectionContainer}>

              <Text style={styles.sectionTitle}>Step One</Text>

              <Text style={styles.sectionDescription}>

                Edit <Text style={styles.highlight}>App.js</Text>

                to change this

                screen and then come back to see your edits.

              </Text>

            </View>

          </View>

        </ScrollView>

      </SafeAreaView>

    </>

  );

};

* 리액트는 HTML 태그를 사용하여 화면을 표시하지만, 리액트 네이티브는 HTML 태그를 사용할 수 없다. HTML 태그 대신 리액트 네이티브에서 정한 특별한 태그(컴포넌트)만 사용
* 리액트 네이티브는 View 컴포넌트를 사용하여 전체적인 레이아웃을 잡는다. 그리고 글자를 표시하기 위해서는 반드시 Text 컴포넌트를 사용해야 한다.
* 상단의 상태 바와 하단에 홈 버튼 영역을 제외한 영역에 콘텐츠 표시: SafeAreaView
* 스크롤 가능 컴포넌트: ScrollView, FlatList, SectionList
* 상단의 상태바 변경: StatusBar

### 스타일링

* 스타일링 방법은 두가지
* StyleSheet.create 함수로 스타일 객체 생성 후, 적용하고 싶은 곳에서 객체 할당
* StyleSheet를 사용하지 않고 직접 스타일 객체를 인라인으로 넣는 방법

1. Style.create

const styles = StyleSheet.create({

footer: {color: Colors.dark, fontSize: 12, fontWeight: '600', padding: 4, paddingRight: 12, textAlign: 'right'},})

<Text style={styles.footer}>Engine: Hermes</Text>

1. 인라인

<Text style={{ fontSize: 24,}}>Engine: Hermes</Text>

* 리액트 네이티브에서의 스타일은 스타일 객체(자바스크립트 객체)이므로 CSS에서 사용되는 요소를 카멜 표기법(Camel Case)를 사용해서 표기

## 추가 라이브러리

### 타입 스크립트(TypeScript)

* 사용 이유
* 자바스크립트는 동적 타이핑 언어로 런타임시 변수의 타입이 결정 🡪 런타임 시 타입이 결정되면 타입 때문에 발생하는 버그와 에러는 실행해야 알 수 있는 점이 문제
* 리액트에서는 플로우(Flow)라는 정적 타입 분석기를 기본적으로 사용.
* 타입스크립트를 권장하는 이유는 자바스크립트 전반에 걸쳐 사용할 수 있기 때문. 플로우보다 좀 더 범용적으로 사용 가능.
* 많은 라이브러리에서 타입스크립트의 타입 정의 파일(DefinitelyTyped)을 제공
* 설치
* npm install typescript @types/react @types/react-native –save-dev
* react-native init FirstApp –template typescript

### Styled Components

* 장점
* React와 React-Native에서 동일한 스타일 코드를 적용할 수 있다
* 웹과 동일한 형태로 스타일을 적용할 수 있다(textAlign 🡪 text-align)
* 단점
* 빌드 후 배포시에 파일 용량이 커짐
* StyleSheet 객체가 생소해질 수도 있음.
* 설치
* npm install –-save styled-components
* npm install –-save-dev @types/styled-components
* Sytled Component를 사용할 때는 Styled.view, Styled.text와 같이 Styled Components가 제공하는 컴포넌트를 사용한다.
* 리액트 네이티브가 제공하는 모든 컴포넌트를 제공.

import Styled from 'styled-components/native';

const Body = Styled.View`

  background-color: ${Colors.white};

`;

const SectionContainer = Styled.View`

  margin-top: 32px;

  padding-horizontal: 24px;

`;

const App = ({}) => {

  return (

    <Fragment>

      <StatusBar barStyle="light-content" />

      <SafeAreaView>

        <ScrollView contentInsetAdjustmentBehavior="automatic">

          <Header />

          <Body>

            <SectionContainer>

              <SectionDescription>Step One</SectionDescription>

              <SectionDescription>Edit <HighLight>App.js</HighLight> to change this screen

            </SectionContainer>

          </Body>

        </ScrollView>

      </SafeAreaView>

    </Fragment>

  );

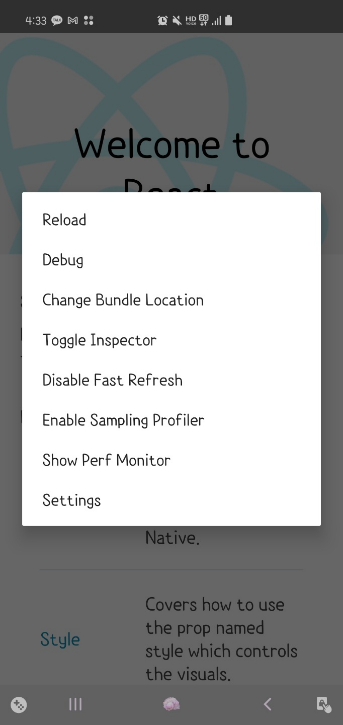
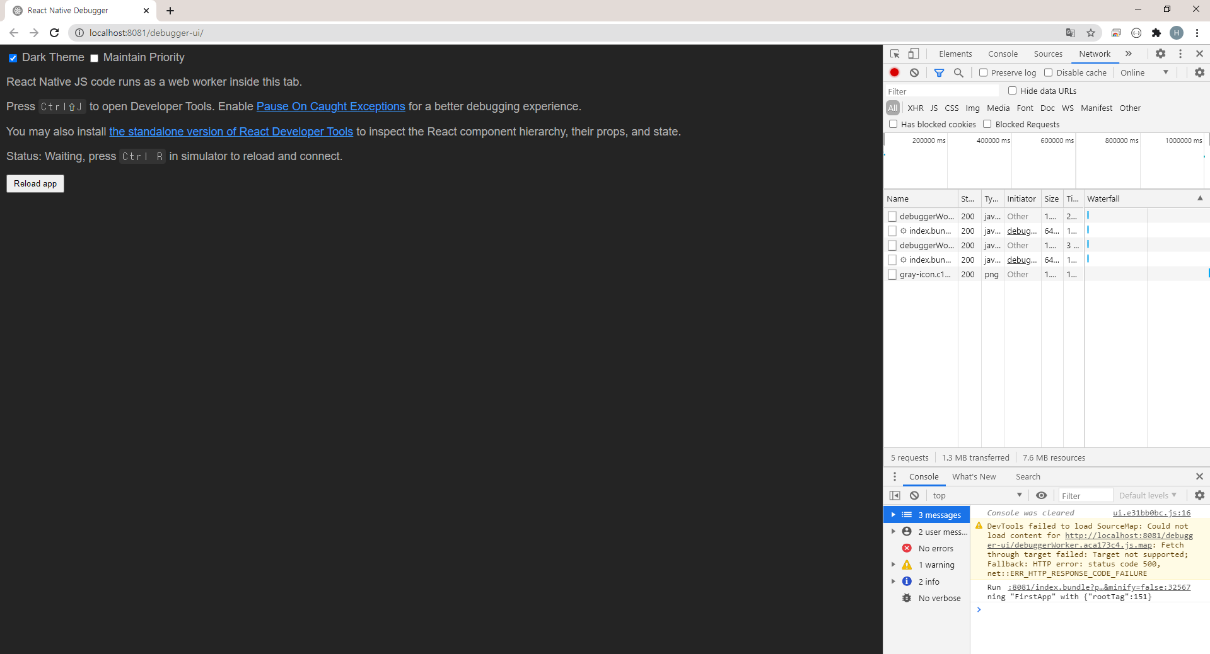
};

### Babel-plugin-root-import

* 사용 이유
* 상대 경로(ex. ‘../../Component’)가 아닌 절대 경로(ex ‘~/Component’)로 사용 가능
* 설치
* npm install --save-dev babel-plugin-root-import

## 개발자 메뉴

* 개발자 모드
* iOS 시뮬레이터의 경우 “Cmd + D”, 안드로이드의 경우 “Cmd + M”을 사용하면 개발자 모드 활성화.
* 실제 단말에서 테스트할 경우, 단말기 흔들기 이벤트(Shake Gesture)로 개발자 메뉴 실행 가능. (개발할 때만 활성화, 앱을 배포하면 사용 불가능)

* 자바스크립트의 디버깅(Debugging)
* 자바스크립트 디버깅: 위의 화면에서 Debug 클릭시
* 패스트 리프레쉬(Fast Refresh)
* React의 핫 리로딩(Hot Reloading), 특별한 설정 X
* iOS에서는 “Cmd + R”, 안드로이드는 “R”키를 연속해서 두번 눌러서 소스코드를 다시 빌드하고 렌더링할 수 있음.

# 카운터 앱(1) – Props와 State

## Props와 State란

* React Native는 React에서 파생되었기 때문에 대부분의 개념이 React와 동일, Props와 Sate 역시 React의 핵심 개념
* Props(Properties): 부모 컴포넌트로부터 자식 컴포넌트로 전달되는 데이터. 부모 컴포넌트로부터 받는 데이터 이므로 자식 컴포넌트에서는 변경이 불가능.
* State: 한 컴포넌트 안에서 유동적인 데이터를 다룰 때 사용되는 상태, 컴포넌트 안에서 데이터를 변경할 수 있음. 즉, 컴포넌트의 상태(state)를 의미함.

## 프로젝트 준비

* 프로젝트 설정 명령
* React-native init Counter
* Cd Counter
* npm install --save styled-components
* npm install --save-dev typescript @types/react @types/react-native @types/styled-components babel-plugin-root-import
* 설정 파일 수정
* tsconfig.json 추가

{

    "compilerOptions": {

        "allowJs": true,

        "allowSyntheticDefaultImports": true,

        "esModuleInterop": true,

        "isolatedModules": true,

        "jsx": "react",

        "lib": ["ES6"],

        "moduleResolution": "node",

        "resolveJsonModule": true,

        "noEmit": true,

        "strict": true,

        "target": "esnext",

        "baseUrl": "./src",

        "paths": {

            "~/\*": ["\*"]

        }

    },

    "exclude": [

        "node\_modules",

        "babel.config.js",

        "metro.config.js",

        "jest.config.js"

    ]

}

* babel.config.js 수정

module.exports = {

  presets: ['module:metro-react-native-babel-preset'],

  plugins: [

    [

      'babel-plugin-root-import',

      {

        rootPathPrefix: '~',

        rootPathSuffix: 'src',

      },

    ],

  ],

};

## 카운터 앱 작성

src 폴더 생성 후, 앱을 작성한다.

### Button 컴포넌트

* src/Component/Button 폴더 생성
* Button/index.tsx 작성
* 참고 소스

interface Props {

  iconName: 'plus' | 'minus';

  onPress?: () => void;

}

const Button = ({iconName, onPress}: Props) => {

  return (

    <Container onPress={onPress}>

      <Icon source={iconName === 'plus'

            ? require('~/Assets/Images/add.png')

            : require('~/Assets/Images/remove.png')

        }

      />

    </Container>

  );

};

export default Button;

* 소스 설명
* 타입스크립트의 interface를 이용하여 컴포넌트의 Props의 타입 지정, 콜론(:) 앞에 ?가 붙으면 필수 X
* Image 컴포넌트에서는 source 속성에 표시하고자 하는 이미지 설정, require 구문을 이용
* Require로 기본 사이즈의 이미지를 연결하고, 기본 사이즈의 이미지 이외에 2x, 3x 크기의 이미지가 있다면, 단말기 화면 사이즈에 맞는 이미지 사이즈를 자동으로 불러와 표시함.

### Counter 컴포넌트

* src/Screens/Counter 폴더 생성
* Counter/index.tsx 작성
* 참고 소스

const Counter = ({title, initValue}: Props) => {

  const [count, setCount] = useState<number>(0);

  return (

    <Container>

      {title && (

        <TitleContainer>

          <TitleLabel>{title}</TitleLabel>

        </TitleContainer>

      )}

      <CountContainer>

        <CountLabel>{initValue + count}</CountLabel>

      </CountContainer>

      <ButtonContainer>

        <Button iconName="plus" onPress={() => setCount(count + 1)} />

        <Button iconName="minus" onPress={() => setCount(count - 1)} />

      </ButtonContainer>

    </Container>

  );

};

export default Counter;

* 소스 설명
* 카운트(Count) 값을 변경하기 위해 이 컴포넌트에서 State를 설정함. 함수형 컴포넌트에서 State를 사용하기 위해서는 리액트 훅의 useState를 사용함.
* const [count, setCount] 여기서 이 카운트 값은 불변(Immutable) 값 🡪 하지만 State는 수정이 가능한 변수라고 했는데, useState 함수는 이 불변의 count 값을 변경하기 위해 set 함수를 같이 제공한다.
* 이런 방식으로 State를 사용하는 이유는, State의 변수 값을 직접 변경하면서 발생할 수 있는 오류를 줄이고 리액트의 가상 돔을 활용하여 변경된 부분만 화면을 갱신하기 위해서임.
* State의 초기 값에 Object 형식의 값도 설정 가능하다.

### App.tsx 수정

* Counter 컴포넌트 이용하여 화면 수정

# 카운터 앱(2) – 클래스 컴포넌트

* 리액트 네이티브는 기존에는 클래스 컴포넌트를 메인으로 사용하였으나, 0.60 버전으로 업데이트되면서 함수형 컴포넌트가 기본 컴포넌트로 변경되었다. 많은 예제들이 클래스형 컴포넌트를 사용하고 있으므로, 클래스 컴포넌트를 다루는 방식을 이해할 필요가 있다.

## 클래스 컴포넌트

### 리액트 훅(React Hooks)

* Hook 이란? 함수형 컴포넌트에서 React state와 생명주기 기능을 “연동(Hook into)”할 수 있게 해주는 것.
* 클래스 컴포넌트를 메인으로 사용한 이유는, 함수형 컴포넌트에서는 State 기능을 사용할 수 없었기 때문이다. 훅 이전에는 State를 가지는 컴포넌트는 클래스 컴포넌트로 제작하고 단순히 Props를 받아 화면에 표시할 때는 함수형 컴포넌트를 사용했다.

## 프로젝트 시작

* 앞의 프로젝트와 대부분 유사하기 때문에 Counter 부분만 다룸.
* Counter/index.tsx 소스 변경
* 컴포넌트 생성 설명
* React의 Component를 상속 받아야함.
* class 형 컴포넌트이기 때문에 생성자(constructor)를 이용해야 하고, 반드시 super 생성자를 호출해야 함.
* 사용할 state가 없는 경우에는 생성자 함수를 만들지 않아도 됨.

interface Props {

  title?: string;

  initValue: number;

}

interface State {

  count: number;

}

class Counter extends React.Component<Props, State> {

  constructor(props: Props) {

    super(props);

    console.log('constructor');

    this.state = {

      count: props.initValue,

    };

  }

* 렌더 함수 설명
* Render() 함수는 화면에 컴포넌트를 렌더링(표시)할 때 호출됨.
* 즉, 이 함수의 반환 값이 화면에 표시
* 클래스 컴포넌트는 함수형 컴포넌트와 다르게 Props와 State에 접근하기 위해서 this를 함께 사용한다.
* 또한, state는 불변 값이므로 this.setState를 이용한다.

…

class Counter extends React.Component<Props, State> {

  …

  render() {

    const {title} = this.props;

    const {count} = this.state;

    return (

      <Container>

        {title && (

          <TitleContainer>

            <TitleLabel>{title}</TitleLabel>

          </TitleContainer>

        )}

        <CountContainer>

          <CountLabel>{count}</CountLabel>

        </CountContainer>

        <ButtonContainer>

          <Button

            iconName="plus"

            onPress={() => this.setState({count: count + 1})}

          />

          <Button

            iconName="minus"

            onPress={() => this.setState({count: count - 1})}

          />

        </ButtonContainer>

      </Container>

    );

  }

}

## 라이프 사이클 함수

* 클래스 컴포넌트는 함수형 컴포넌트와 다르게 라이프사이클 함수들을 가지고 있다. 이 라이프사이클을 이해하면 클래스 컴포넌트를 좀 더 효율적으로 활용할 수 있다.

### Constructor 함수

* 사용 목적: 컴포넌트의 state를 초기화
* 호출 시점: 해당 컴포넌트가 생성될 때 한번만 호출
* 부연 설명
* State를 사용하지 않아, State의 초기 값 설정이 필요하지 않으면, 생성자 함수도 생략 가능
* 생성자 함수를 사용할 때는 반드시 super(props) 함수를 호출하여 부모 클래스의 생성자를 호출.

### Render 함수

* 사용 목적: 클래스 컴포넌트가 렌더링되는 부분(화면에 표시되는 부분)을 정의
* 호출 시점
* 부모로부터 받은 Props 값이 변경될 때 호출
* this.setState로 State 값이 변경되어 화면을 갱신할 필요가 있을 때마다 호출
* 부연 설명
* 이 함수에서 this.setState를 사용하여 State 값을 직접 변경할 경우, 무한루프에 빠질 수 있으므로 주의
* 이벤트에 트리거를 걸면 상관 X

### getDerivedStateFromProps 함수

* 사용 목적: 부모로부터 받은 Props와 State를 동기화 할 때 사용
* 다시 말해, 부모로부터 받은 Props로 State 값을 설정하거나, Props에 의존하여 State 값을 결정하고자 할 때 이 함수를 사용
* 호출 시점
* 컴포넌트가 생성될 때 한 번 호출
* Props와 State를 동기화 해야 하므로 Props가 변경될 때 마다 호출(생성자 함수와 다른 점)
* 예제 소스

static getDerivedStateFromProps(nextProps, prevState){

    if(nextProps.id !== prevState.id){

      return {value: nextProps.value};

    }

    return null

  }

* State에 설정하고 싶은 값을 이 함수에서 반환하게 되며, 동기화할 State가 없으면 null 반환

### componentDidMount 함수

* 호출 시점: 클래스 컴포넌트가 처음으로 화면에 표시된 이후, 이 함수가 딱 한번만 호출됨.
* 한번만 호출되므로 ajax를 통해 데이터를 습득 및 다른 자바스크립트 라이브러리와의 연동 수행에 적합.
* 이 함수는 Props 값이나 State 값이 변경되어도 다시 호출하지 않으므로, this.setState를 직접 호출할 수 있으며, ajax를 통해 받은 데이터를 State에 설정하기에 적합하다.

### shouldComponentUpdate 함수

* 사용 목적: Props나 State가 변경될 때 다시 화면을 그리고 싶지 않은 경우에 이 함수를 이용하여 렌더링을 제어
* 이 함수를 이용하여 데이터를 비교하고 불필요한 리렌더링을 방지하면 좀 더 성능이 좋은 앱을 제작할 수 있음.
* 예제 소스

  shouldComponentUpdate(nextProps, nextState){

    console.log('shouldComponentUpdate');

    return nextProps.id !=== this.props.id;

  }

* False를 반환하면 리렌더링 금지

### getSnapshotBeforeUpdate 함수

* 호출 시점: Props 또는 State가 변경되어 화면을 다시 그리기 위해 render 함수가 호출된 후, 실제로 화면이 갱신되기 바로 직전에 호출.
* 부연 설명
* 이 함수의 반환 값(snapshot)은 componentDidUpdate의 세번째 매개변수로 전달됨.
* 많이 활용되지는 않지만, 화면을 갱신하는 동안 수동으로 스크롤 위치를 고정하는 등에 사용될 수 있음.
* Warning
  + getSnapshotBeforeUpdate 선언 후 반환 값을 반환하지 않는 경우
  + getSnapshotBeforeUpdate를 선언하고 componentDidUpdate를 선언하지 않은 경우

### ComponentDidUpdate 함수

* 호출 시점: ComponentDidMount 함수(컴포넌트가 첫 렌더링 됐을 때만 호출)와는 반대로 컴포넌트가 첫 렌더링 됐을 때에는 실행되지 않지만, Props와 State가 변경되어 화면이 갱신될 때마다 render함수 호출 이후에 호출
* 부연 설명
* render 함수와 마찬가지로 이 함수는 State 값이 변경될 때마다 호출되므로 setState를 직접 호출하면 무한루프에 빠질 수 있음.

### comonentWillUnmount 함수

* 사용 목적: 할당된 리소스 해제
* 연동한 자바스크립트 라이브러리 해지
* setTimeout, setInterval 등의 타이머를 clearTimeout, clearInterval을 사용하여 해제할 때 사용.
* 호출 시점: 클래스 컴포넌트가 화면에서 완전히 사라진 후 호출
* 클래스 컴포넌트가 사라진 후이기 때문에 setState를 호출하면 Warning이나 메모리 누수(Memory leaks)가 발생할 수 있음.

### componentDidCatch 함수

* 사용 목적: 에러의 예외 처리 함수
* 호출 시점: return 부분에서 에러 발생 시 호출
* 예제 소스

interface State {

  count: number;

  error: boolean;

}

class Counter extends React.Component<Props, State> {

  constructor(props: Props) {

    super(props);

    this.state = {

      count: props.initValue,

      error: false,

    };

  }

  render() {

    const {title} = this.props;

    const {count, error} = this.state;

    return (

      <Container>

        {!error && ()}

      </Container>

  componentDidCatch(error: Error, info: React.ErrorInfo) {

    this.setState({

      error: true,

    });

  }

}

### 호출 순서

1. 컴포넌트가 생성될 때

constructor 🡪 getDerivedStateFromProps 🡪 render 🡪 componentDidMount

1. 컴포넌트의 Props가 변경될 때

getDerivedStateFromProps 🡪 shoudComponentUpdate 🡪 render 🡪 getSnapshotBeforeUpdate 🡪 componentDidUpdate

1. 컴포넌트의 State가 변경될 때

showComponentUpdate 🡪 render 🡪 getSnapshotBeforeUpdate 🡪 componentDidUpdate

1. 컴포넌트 렌더링 중 에러가 발생될 때: componentDidCatch
2. 컴포넌트가 제거될 때: componentWillUnmount

# TodoList App – Context와 AsyncStorage

* React에서 데이터를 다루는 방법으로 Props와 State, 그리고 Context가 존재
* 이 앱을 통해 Context API를 다루는 방법에 대해서 실습하고, 앱 내에 데이터를 저장하는 방법인 AsyncStorage에 대해 실습

## Context API