230612_실습

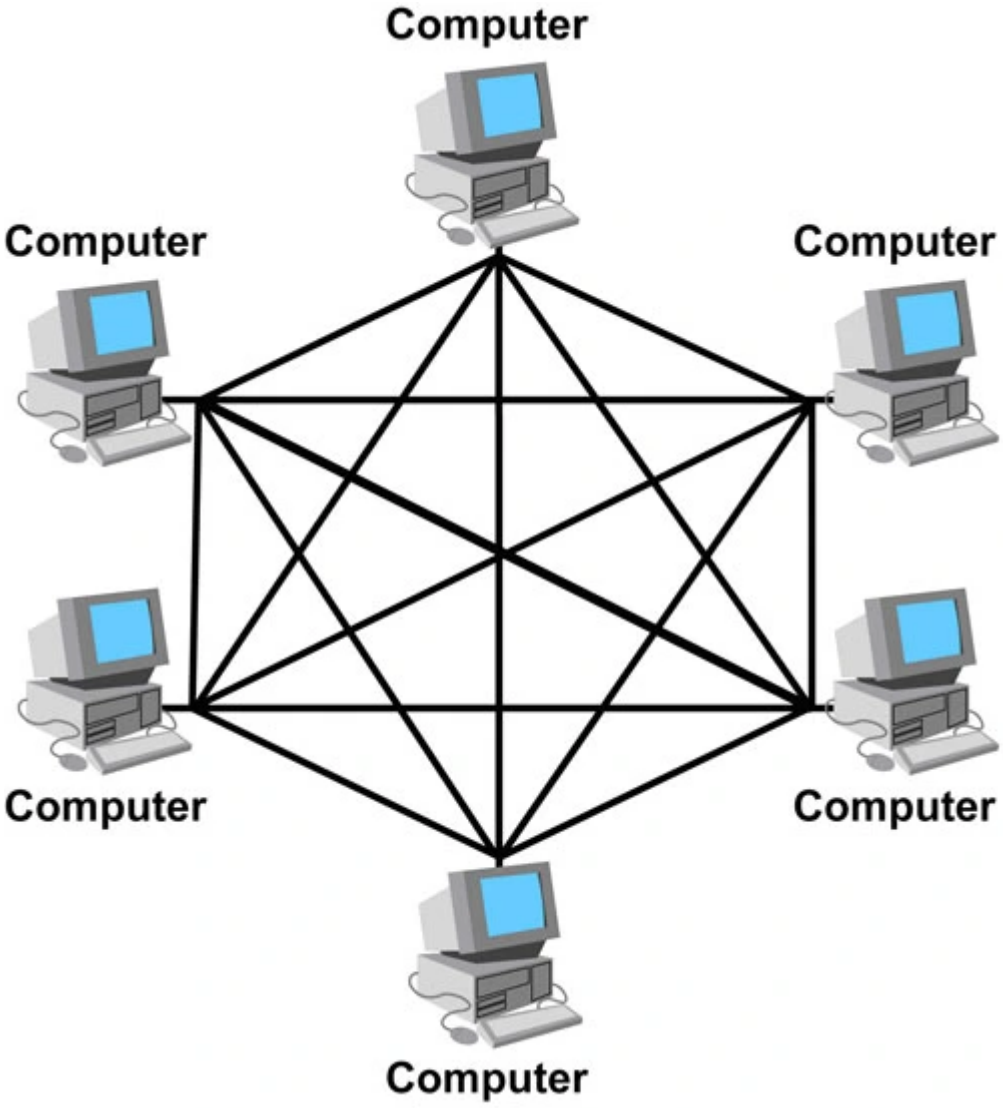
Node.js

Node.js 개요

Node.js 란?

💡 **Node**

- 노드(node)는 **컴퓨터 과학에 쓰이는 기초적인 단위**
- 변과 함께 그래프를 구성하는 요소. 결절(結節), 정점(頂點), 점 등이라 한다. 그래프를 이루는 점과 선 중에서 점을 ***노드*** 또는 절점이라 한다.
- **노드는 정보를 전송, 수신 및 / 또는 전달할 수있는 다른 장치의 네트워크에있는 모든 물리적 장치입니다.**



- Node.js는 Chrome V8 JavaScript 엔진으로 빌드 된 JavaScript 런타임입니다.
- 노드를 통해 **다양한 자바스크립트 애플리케이션을 실행**할 수 있으며, 서버를 실행하는 데 주로 사용
- Node.js는 JavaScript를 서버에서도 사용할 수 있도록 만든 프로그램
- Node.js는 V8이라는 JavaScript 엔진 위에서 동작하는 자바스크립트 런타임(환경)
- Node.js는 서버사이트 스크립트 언어가 아닌 프로그램(환경)
- Node.js는 웹서버와 같이 확장성 있는 네트워크 프로그램 제작이 목적
- 내장 HTTP 서버 라이브러리를 포함하고 있어 웹 서버에서 **아파치 등의 별도 소프트웨어 없이 동작하는 것이 가능,** 이를 통한 웹 서버의 동작에 있어 더 많은 통제에서 벗어나 여러 가지 기능이 가능

Node.js 사용 이유

- JavaScript 를 웹 브라우저에서 독립시킨 것으로 Node.js를 설치하게 되면 터미널프로그램(윈도우의 cmd, 맥의 terminal 등)에서 Node.js를 입력하여 브라우저 없이 바로 실행할 수 있다.
- Node.js를 이용하여 웹 브라우저와 무관한 프로그램을 만들 수 있다.
- Node.js를 이용하여 **서버를 만들 수 있다.**
- 이전까지 Server-Client 웹사이트를 만들 때 웹에서 표시되는 부분은 JavaScript 를 사용하여 만들어야만 했으며, 서버는 Reby, Java 등 다른 언어를 써서 만들었어야 했는데 마침내 **한 가지 언어로 전체 웹 페이지를 만들 수 있게 된 것**이다.
- 이벤트 기반, 논 블로킹 I/O 모델을 사용해 가볍고 효율적

Node.js 특징

- **비동기 I/O 처리 / 이벤트 위주:** Node.js 라이브러리의 모든 API는 비동기식입니다, 멈추지 않는다는거죠 (Non-blocking). Node.js 기반 서버는 API가 실행되었을때, 데이터를 반환할때까지 기다리지 않고 다음 API 를 실행합니다. 그리고 이전에 실행했던 API가 결과값을 반환할 시, NodeJS의 이벤트 알림 메커니즘을 통해 결과값을 받아옵니다.
- **빠른 속도:** 구글 크롬의 V8 자바스크립트 엔진을 사용하여 빠른 코드 실행을 제공합니다.
- **단일 쓰레드 / 뛰어난 확장성:** Node.js는 이벤트 루프와 함께 단일 쓰레드 모델을 사용합니다. 이벤트 메커니즘은 서버가 멈추지않고 반응하도록 해주어 서버의 확장성을 키워줍니다. 반면, 일반적인 웹서버는 (Apache) 요청을 처리하기 위하여 제한된 쓰레드를 생성합니다. Node.js 는 쓰레드를 한개만 사용하고 Apache 같은 웹서버보다 훨씬 많은 요청을 처리할 수 있습니다.
- **노 버퍼링:** Node.js 어플리케이션엔 데이터 버퍼링이 없고, 데이터를 chunk로 출력합니다.
- **라이센스:** Node.js 는 MIT License가 적용되어있습니다.

Node.js 설치

1. 직접 설치

> 💡 node.js 공식 홈페이지
> https://nodejs.org/en/

- LTS(Long Term Supported)

    - 장기적으로 안정되고 신뢰도가 높은 지원이 보장되는 버전
    - 유지/보수와 보안(서버 운영 등)에 초점을 맞춰 대부분 사용자에게 추천되는 버전
    - **짝수 버전(ex. 8.x.x)이 LTS 버전**
- Current(현재 버전)
    - 최신 기능을 제공하고 기존 API의 기능 개선에 초점이 맞춰진 버전
    - 업데이트가 잦고 기능이 변경될 가능성이 높기 때문에 간단한 개발 및 테스트에 적당한 버전
    - **홀수 버전(ex. 9.x.x)이 Current 버전**
- 설치 확인

```
node -v
# v8.9.4
npm -v
# 5.6.0
```

    - 설치 후 버전 확인 명령을 통해 node명령이 정상적으로 동작하는지 확인

2. 패키지 매니저로 설치
    - macOS - Homebrew

```
brew install node@8
```

        - 버전의 Major Number만 입력
    - Windows - Chocolatey

```
choco install nodejs-lts
```

NPM(Node Package Manager)

NPM 이란?

- Node.js 개발자들이 패키지(모듈)의 설치 및 관리를 쉽게 하기 위해 도와주는 매니저(관리 도구)

💡 패키지(모듈) : 프로그램의 구성요소 중 특정 기능을 수행할 수 있는 코드의 집합(라이브러리).

유명한 플랫폼(프로그래밍 언어, OS 등)은 저마다의 패키지 매니저를 가지고 있는데,

다른 유명 패키지 매니저로는

- **Python**의 pip
- **Java**의 Maven, Gradle(Android, React Native에서 많이 봤는데??? 그거 맞습니다.^^)
- **PHP**의 Composer
- **Ruby**의 RubyGems

등이 있고,

Linux환경에 익숙하신 분들은

- **레드햇 계열**의 rpm, yum
- **데비안 계열**의 dpkg, apt
- **맥 OS의** Homebrew

등이 있습니다.

- npm은 JavaScript 및 세계 최대의 소프트웨어 레지스트리 패키지 관리자
- node.js 설치시 같이 설치됨
- npm에는 Node.js에서 사용되는 각종 코드 패키지들이 모여있고, 우리는 그 패키지를 다운로드 받아 사용할 수 있습니다.
- 쉽게 npm은 Node.js 생태계의 앱스토어나 플레이스토어 같은 역할
- npm 레지스트리에는 640,000개가 넘는 패키지가 포함
- 패키지는 의존성(dependencies) 및 버전을 추적할 수 있도록 구성

💡 npm
https://www.npmjs.com/

NPM 장점

- 프로그램을 제작 시 어떤 기능을 구현할 때 자신이 직접 프로그래밍을 하지 않아도 동일한 기능의 남이 만들어놓은 코드를 쉽게 사용이 가능하다.
- 코드의 재사용성이 높아지고 유지 보수가 쉬워질뿐더러 형상관리가 용이해진다.

NVM(Node Version Manager)

NVM 이란?

- Node.js 의 버전을 관리하는 도구
- NVM을 사용하는 이유
  - **협업을 할 때, 또는 다양한 프로젝트를 동시에 진행해야 할 때**
  - 다양한 라이브러리 / 프레임워크 / 개발툴의 버전 호환 문제발생
- NVM의 장점
  - 다양한 버전의 Node.js를 설치 가능
  - 명령을 통해 다양한 Node 버전으로 스위칭 가능
  - 디폴트 버전을 설정하거나 / 설치한 버전들의 전체 리스트를 확인하거나 / 필요 없는 버전을 삭제하는 등 버전 관리가 용이
  - `Ruby`의 `rvm`, `rbenv`나 `Python`의 `pyenv`도 같은 역할

node.js 명령어

npm 명령어

1. 버전 확인

```
npm -version
#또는
npm -v
```

2. node.js 프로젝트 시작

```
npm init
```

- Node.js 프로젝트를 시작할때 package.json을 생성해 주는 명령

`package.json`
프로젝트의 정보와 특히 프로젝트가 의존하고 있는(설치한) 패키지(모듈)에 대한 정보가 저장되어 있는 파일.

- package.json 예
  - 기본 package.json

```
{
  "name": "example",
  "version": "1.0.0",
  "description": "example",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "repository": {
    "type": "git",
    "url": "example"
  },
  "keywords": [
    "example"
  ],
  "author": "minchan",
  "license": "MIT"
}
```

  - react native

```
{
  "name": "example",
  "version": "0.0.1",
  "private": true,
  "scripts": {
    "android": "react-native run-android",
    "ios": "react-native run-ios",
    "start": "react-native start",
    "test": "jest",
    "lint": "eslint ."
  },
  "dependencies": {
    "@react-native-community/masked-view": "0.1.6",
    "@react-navigation/bottom-tabs": "5.0.0",
    "@react-navigation/drawer": "5.0.0",
    "@react-navigation/native": "5.0.0",
    "@react-navigation/stack": "5.0.0",
    "react": "16.9.0",
    "react-native": "0.61.5",
    "react-native-gesture-handler": "1.5.6",
```

```
    "react-native-reanimated": "1.7.0",
    "react-native-safe-area-context": "0.7.0",
    "react-native-screens": "2.0.0-beta.2"
  },
  "devDependencies": {
    "@babel/core": "7.8.4",
    "@babel/runtime": "7.8.4",
    "@react-native-community/eslint-config": "0.0.7",
    "@types/react": "16.9.19",
    "@types/react-native": "0.61.10",
    "babel-jest": "25.1.0",
    "eslint": "6.8.0",
    "jest": "25.1.0",
    "metro-react-native-babel-preset": "0.58.0",
    "react-test-renderer": "16.9.0",
    "typescript": "3.7.5"
  },
  "jest": {
    "preset": "react-native"
  }
}
```

- 이렇게 package.json에 **프로젝트에 대한 정보**, npm과 연결하여 **사용할 명령**("`scripts`"객체), 프로젝트가 의존하고 있는(설치한) **패키지(모듈)에 대한 정보**("`dependencies`"객체), 프로젝트의 **개발과 관련된 테스트, 컴파일, 코드 작성 형태와 같은 패키지(모듈)에 대한 정보**("`devDependencies`"객체) 등이 저장되어 있는 것을 확인하실 수 있습니다.

3. 패키지 설치
   - 문법

```
npm install (option) [package]
```

   - 옵션
     - `g` : 패키지가 해당 프로젝트(local)가 아닌 시스템 레벨에 전역(global) 설치되어 다른 Node.js 프로젝트에서도 사용할 수 있게 됩니다.
     - `-save`(`-S`) : package.json의 "dependencies"객체에 추가됩니다. (**npm5부터 default**로 설정되어 더 이상 사용하지 않습니다.)
     - `-save-dev`(`-D`) : package.json의 "devDependencies"객체에 추가됩니다.
     - **@패키지 버전** : 패키지명 뒤에 @패키지 버전을 쓰시면 해당 버전의 패키지가 설치되며 입력하지 않을 시 최신 버전으로 설치가 됩니다.

> 💡 **npm install(npm i) 사용팁**
>
> 만약 패키지명을 입력하지 않고 npm install(npm i)만 입력할 시 package.json의 "dependencies"객체에 명시되어 있는 패키지(모듈)들을 모두 설치하게 됩니다.
>
> 따라서 프로젝트의 형상관리를 위해 `GitHub`와 같은 저장소에 업로드할 때, 무겁고 수많은 패키지(모듈)들을 전부 업로드하는 것이 아니라, `package.json`만 업로드해놓으시면 나중에 프로젝트를 내려받았을 때 npm i 명령어를 통해서 기존 프로젝트에서 사용하던 패키지(모듈)들을 손쉽게 원상복귀시키실 수 있습니다!
>
> (.gitignore파일에 node_modules를 추가하여 패키지(모듈)들이 commit되지 않게 설정해두세요.)

4. 패키지 삭제
   - 문법

```
npm uninstall [package]
```

   - npm uninstall 명령 뒤에 삭제하실 패키지명을 입력하시면 설치된 패키지가 node_modules폴더에서 삭제될 뿐만 아니라 package.json의 "dependencies"객체에서도 삭제됩니다.
   - **단, 설치하실 때 옵션을 사용하셨다면 삭제하실 때도 같은 옵션을 넣어주세요.**
5. 패키지 업데이트
   - 문법

```
npm update [package]
```

   - 설치된 패키지를 최신 버전으로 업데이트합니다.
   - 의존성이 엮여있는 패키지를 함부로 업데이트하면 잘 돌아가던 프로젝트가 갑자기 에러의 굴레에 휘말릴 수 있으니 기존의 버전을 기억해두시거나 신중하게 업데이트 하셔야 합니다.
6. 초기화
   - 명령어

```
npm cache clean
npm rebuild
```

- 두 가지 명령을 차례로 명령해주시면 됩니다.

> 💡 npm cache clean 명령은 npm의 cache를 지우는 명령이고 npm rebuild 명령은 npm을 새롭게 재설치 하는 명령입니다. 주로 npm 명령어가 안 먹히거나 기타 잡다한 버그가 생겼을 시 해결하기 위한 조치 방법으로 쓰이니, 꼭 기억해 두셨다가 npm에 문제가 발생했을 때 사용해보시면 좋을 것 같습니다!

7. `package.json`

- `npm init` 명령어로 초기화를 해주면 `package.json` 파일이 자동으로 생성되는데, 이 때 `package.json` 파일

```
{
  "name": "sample",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
```

- `scripts`
  - shell script를 지정해 두면 필요할 때 실행 가능
  - `test` 라는 스크립트가 지정되어 있는데, `npm test` 를 입력하여 해당 스크립트를 실행하면 다음과 같은 메세지가 출력되는 것을 확인

```
npm test
#Error: no test specified\" && exit 1
```

  - 예제

```
#만약 새로운 스크립트를 지정하고 싶다면 다음과 같이 새로운 라인에 스크립트를 추가하고 실행
"scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "msg": "echo \"Hello, World\""
},
```

```
npm run msg
```

nvm 명령어

1. 특정 버전의 node.js 설치
   - 문법

```
nvm install [version]
```

   - 예제

```

# node.js 버전 설치하기
nvm install 0.10
nvm install v0.1.2
nvm install v8

# node 최신 버전 설치 (설치 당시 기준)
nvm install node

# node LTS 최신버전 설치
nvm install --lts
```

2. 설치된 버전 확인 및 삭제

```
# 설치된 node.js 목록 확인하기
nvm ls

# 설치할 수 있는 모든 Node 버전 조회 (재미삼아 해보지마세요 겁나많음... 황급히 control C 두드리기)
$ nvm ls-remote

# 특정 버전의 node 사용하기
$ nvm use <version>

# 현재 사용중인 버전 확인하기
$ nvm current

# node.js 설치 경로 확인하기
```

```
$ which node

# 필요없는 node 버전 삭제하기
$ nvm uninstall <version>
```

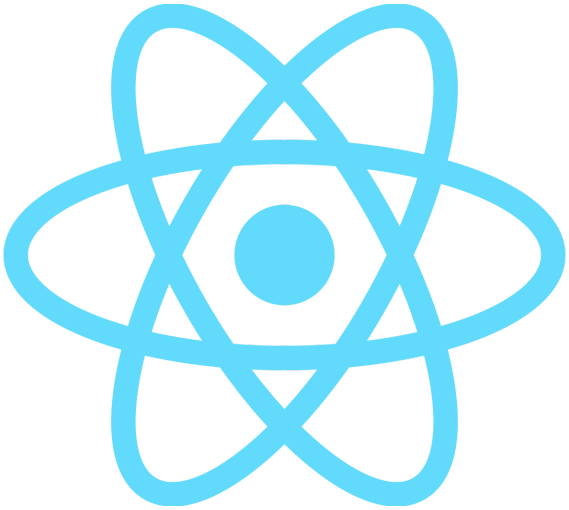
3. 기본 버전 설정

```
$ nvm alias default 8.9.4

# 설치되어 있는 가장 최신버전의 node를 디폴트로 사용하기
$ nvm alias default node
```

- 만일 새로운 쉘을 실행할 경우 `node` 의 버전이 `system` 버전으로 리셋되는데요, 이를 고정하기 위한 커맨드는 다음과 같습니다.

React



React 개요

React 란?

- 2013년 페이스북(현 메타)에서 개발
- 자바스크립트 라이브러리
- 사용자 인터페이스를 만들기 위해 사용
- 지속적으로 데이터가 변화하는 대규모 애플리케이션을 구축하는 것을 목표

Front-end 기술의 세대별 분류

1. 1세대 기술
    - HTML / CSS
    - JavaScript
    - DOM
    - Event
2. 2세대 기술
    - jQuery
3. 3세대 기술
    - Angular(Framework)
    - Vue(Framework)
    - React(Library)

준비

react 설치

```
// react app을 생성할 수 있는 명령어 create-react-app을 설치한 후
$ npm install -g create-react-app
```

설치 확인

```
// create-react-app 버전을 확인합니다.
$ create-react-app --version
```

react 프로젝트 생성

```
// hello-react라는 이름의 react app을 생성합니다.
$ create-react-app hello-react
```

react app 실행

```
// 이 react app을 실행합니다.
$ npm start
```

PORT 설정

1. 명령어로 포트 설정하기

```
PORT=3001 npm start
```

2. package.json 파일 설정 수정하기
   - 1. React 프로젝트 폴더에 ".package.json" 파일을 찾아 아래와 같이 수정한다.
   - 2. React 앱을 실행시킨다. ex) npm start
   - Mac OS, LINUX 환경

```
"scripts": {
  "start": "export PORT=3001 && react-scripts start",
  ...
}
```

   - Windows 환경

```
"scripts": {
  "start": "set PORT=3001 && react-scripts start",
  ...
}
```

3. ".env" 파일 만들기
   - 1. React 프로젝트 폴더에 ".env" 파일 만든다.
   - 2. ".env" 파일에 아래와 같이 "PORT=사용할포트" 형식으로 값을 입력한 후 저장한다.
   - 3. React 앱을 실행시킨다. ex) npm start
   - .env

```
PORT=3001
```

React 자습서

➡️ https://ko.legacy.reactjs.org/tutorial/tutorial.html

JSX

JSX 소개

JSX란?

```
const element = <h1>Hello, world!</h1>;j
```

```
// 실제 작성할 JSX 예시
function App() {
  return (
      <h1>Hello, GodDaeHee!</h1>
    );
}

// 위와 같이 작성하면, 바벨이 다음과 같이 자바스크립트로 해석하여 준다.
function App() {
  return React.createElement("h1", null, "Hello, GodDaeHee!");
}
```

- JSX(JavaScript XML)는 Javascript에 XML을 추가하여 확장한 문법이다.
- JSX는 리액트로 프로젝트를 개발할 때 사용되므로 공식적인 자바스크립트 문법은 아니다.
- JSX는 **React "엘리먼트(element)"** 를 생성한다. React 엘리먼트는 브라우저 DOM 엘리먼트와 달리 일반 **객체**이다.
- React는 JSX 사용이 필수가 아니지만, JS 코드 안에서 UI관련 작업을 할 수 있기 때문에 시각적으로 더 도움이 된다. 또한 JSX를 사용하면 React가 더욱 도움이 되는 에러 및 경고 메시지를 표시할 수 있게 해준다.
- 장점
  1. 보기 쉽고 익숙하다.
     - JSX는 HTML 코드와 비슷하기 때문에 일반 자바스크립만 사용한 코드보다 더 익숙하며 가독성이 좋다.
  2. 높은 활용도

- JSX에는 div, span 같은 HTML 태그를 사용할 수 있으며, 개발자가 만든 컴포넌트도 JSX 안에서 작성할 수 있다.

JSX 사용법

표현식

- JSX 안에 자바스크립트 표현식 을 중괄호로 묶어서 포함시킬 수 있습니다.
- 자바스크립트 표현식을 작성하려면 JSX내부에서 코드를 { }로 감싸주면 된다.

```
function App() {
  const name = 'Snoopy';
  return (
    <div>
      <div>Hello</div>
      <div>{name}!</div>
    </div>
  );
}
```

```
const name = "홍길동";
const nameDetail = {
    firstName: "홍",
    lastName: "길동",
};

const element = <h1>성은 {nameDetail.firstName}이요,
이름은 {nameDetail.lastName}, 전체 이름은 {name}</h1>;

ReactDOM.render(element, document.getElementById("root"));
```

닫힌 코드

- **반드시 부모 요소 하나가 감싸는 형태여야 한다.**
  - Virtual DOM에서 컴포넌트 변화를 감지할 때 때 효율적으로 비교할 수 있도록 컴포넌트 내부는 하나의 DOM 트리 구조로 이루어져야 한다는 규칙이 있기 때문이다.
  - 잘못된 코드
    - 예제1

```
function App() {
  return (
    <div>Hello</div>
    <div>World!</div>
```

```
  );
}
```

- 예제2

```
import React from 'react';
import Hello from './Hello';

function App() {
  return (
    <Hello />
    <div>안녕히계세요.</div>
  );
}

export default App;
```

- 바른 코드
  - 예제1

```
// div를 사용 하였기 때문에 스타일 적용시 작성한 코드를 div로 한번 더 감쌌다는 부분을 고려해야
 한다.
function App() {
  return (
    <div>
      <div>Hello</div>
      <div>World!</div>
    </div>
  );
}
```

```
function App() {
  return (
    <Fragment>
      <div>Hello</div>
      <div>World!</div>
    </Fragment>
  );
}
```

```
function App() {
  return (
    <>
      <div>Hello</div>
      <div>World!</div>
    </>
```

```
  );
}
```

- 예제2

```
import React from 'react';
import Hello from './Hello';

function App() {
  return (
    <div>
      <Hello />
      <div>안녕히계세요</div>
    </div>
  );
}

export default App;
```

```
import React from 'react';
import Hello from './Hello';

function App() {
  return (
    <>
      <Hello />
      <div>안녕히계세요</div>
    </>
  );
}

export default App;
```

- `<div>`, `<p>`, `<span>`, `<a>` 같이 짝이 있는 태그의 경우 반드시 닫는 태그가 존재해야 한다. 그렇지 않을 경우 에러가 발생한다.
- `<img/>`, `<input/>`, `<br/>` 같은 단독 태그(self-closing tag)의 경우에는 반드시 태그를 닫아줘야 한다. 그렇지 않을 경우 에러가 발생한다.
  - 잘못된 코드

```
import React from 'react';
import Hello from './Hello';

function App() {
  return (
    <div>
      <Hello />
      <Hello />
```

```
      <Hello />
      <div>
    </div>
  );
}

export default App;
```

```
import React from 'react';
import Hello from './Hello';

function App() {
  return (
    <div>
      <Hello />
      <Hello />
      <Hello />
      <input>
      <br>
    </div>
  );
}

export default App;
```

- 바른 코드

```
import React from 'react';
import Hello from './Hello';

function App() {
  return (
    <div>
      <Hello />
      <Hello />
      <Hello />
      <input />
      <br />
    </div>
  );
}

export default App;
```

style과 className

- 일반 HTML에서 CSS 클래스를 사용할 때에는 class 라는 속성을 사용한다.
- JSX에서는 class가 아닌 className 을 사용한다.

- jsx에서 style를 작성할 때로 "-"가 아니라 cameICase를 사용해야 합니다
  - 예제1

```
function App() {
    return (
        <>
            <h1 class="abc">{title}</h1> <--// X 오류 발생
            <h1 className="abc">{title}</h1> <--// O 정상 작성
            <Desc />
            <Body />
        </>
    );
}
```

  - 예제2

App.css

```
.gray-box {
  background: gray;
  width: 64px;
  height: 64px;
}
```

App.js

```
import React from 'react';
import Hello from './Hello';
import './App.css';


function App() {
  const name = 'react';
  const style = {
    backgroundColor: 'black',
    color: 'aqua',
    fontSize: 24, // 기본 단위 px
    padding: '1rem' // 다른 단위 사용 시 문자열로 설정
  }

  return (
    <>
      <Hello />
      <div style={style}>{name}</div>
      <div className="gray-box"></div>
    </>
  );
}
```

```
export default App;
```

- style 속성을 사용한다.
  - 예제

```
import React from 'react';
import Hello from './Hello';

function App() {
  const name = 'react';
  const style = {
    backgroundColor: 'black',
    color: 'aqua',
    fontSize: 24, // 기본 단위 px
    padding: '1rem' // 다른 단위 사용 시 문자열로 설정
  }

  return (
    <>
      <Hello />
      <div style={style}>{name}</div>
    </>
  );
}

export default App;
```

주석

- JSX 내에서 {/…/} 와 같은 형식을 사용 한다.

```
function App() {
  return (
    <>
      {/* 주석사용방법 */}
      <div>Hello, GodDaeHee!</div>
    </>
  );
}
```

- 시작태그를 여러줄 작성시에는, 내부에서 // 의 형식을 사용할 수 있다.

```
function App() {
  return (
    <>
      <div
      // 주석사용방법
```

```
      >Hello, GodDaeHee!</div>
    </>
  );
}
```

Function Component

- 다양한 Function 형태로 컴포넌트화할 수 있습니다.

```
const title = "기능 컴포넌트화하기";
function Desc() {
    return (
        <h2>Function 형태로 컴포넌트화할 수 있습니다.</h2>
    );
}
const Body = () => <p> 본문 내용이 나오는 영역입니다. </p>;
const Footer = () => {return <p> Footer 영역입니다.</p>;};

// 함수형 컴포넌트
function App() {
    return (
        <>
            <h1>{title}</h1>
            <Desc></Desc>
            <Body />
            <Footer />
        </>
    );
}

ReactDOM.render(<App />, document.getElementById("root"));
```

Class Component

- Class 형태로도 컴포넌트화할 수 있습니다.

```
const title = "기능 컴포넌트화하기";
function Desc() {
    return (
        <h2>Function 형태로 컴포넌트화할 수 있습니다.</h2>
    );
}
const Body = () => <p> 본문 내용이 나오는 영역입니다. </p>;
const Footer = () => {return <p> Footer 영역입니다.</p>;};

// 클래스 형 컴포넌트
class App extends React.Component {
    render() {
        return (
          <>
            <h1>{title}</h1>
```

```
            <Desc></Desc>
            <Body />
            <Footer />
          </>
        );
    }
}

ReactDOM.render(<App />, document.getElementById("root"));
```

Props 사용하기

- Props를 이용하여 컴포넌트에 값을 넘길 수 있습니다.

```
const Body = (props) => {
    return
        <>
          <p> 제목 : {props.title}</p>;
          <p> 내용 : {props.desc}</p>;
        </>
};

function App() {
    return (
        <Body title="제목입니다." desc="설명입니다." />
    );
}
```

구조 분해 할당 (튜플 분해)

- props를 넘길 때 구조를 분해해서 할당할 수도 있습니다.

```
const Body = ({ title, desc }) => {
    return
        <>
            <p> 제목 : {title}</p>;
            <p> 내용 : {desc}</p>;
        </>
};

function App() {
    return (
        <>
          <Body title="제목입니다." desc="설명입니다." />
        </>
    );
}
```

조건(3항) 연산자 사용하기

- jsx의 표현식을 활용할 때 if 문은 사용할 수 없고 대신에 3항 연산자로 사용할 수 있습니다. (3항 연산자란 일반적인 if 문을 축약한 형태입니다.)
- JSX 내부의 JS 표현식에서는 if문을 사용할 수 없다. 때문에 조건에 따라 다른 내용을 렌더링 하고자 할 경우 JSX 밖에서 if 문을 사용하거나, 중괄호 안에서 삼항 연산자를 사용하면 된다.

```
// 일반 if 문.
if (yn === true) {
    return <Body title="제목입니다." desc="설명입니다." />;
} else {
    return <Body title="없습니다." desc="아닙니다." />;
}

// 3항 연산자
{
    yn === true ? <Body title="제목입니다." desc="설명입니다." /> : <Body title="없습니다." d
esc="아닙니다." />;
}

// AND 연산자 : true 면 실행, false 면 아무것도 없음.
{
    yn === true && <Body title="제목입니다." desc="설명입니다." />;
```

```
class App extends Component {
    render() {
        let name = 'React';
        return (
            <div>
                {
                    name === 'React' ? (
                        <h1>This is React.</h1>
                    ) : (
                        <h1>This is not React.</h1>
                    )
                }
            </div>
        );
    }
}
```

- AND 연산자(&&)를 사용한 조건부 렌더링
  - 특정 조건을 만족할 때만 내용을 보여주고 싶을 때 사용

```
class App extends Component {
    render() {
        let name = 'React';
        return (
```

```
            <div>
                {
                    name === 'React' && <h1>This is React.</h1>
                }
            </div>
        );
    }
}
```

- OR 연산자(||)를 사용한 조건부 렌더링
  - 리액트 컴포넌트에서는 함수에서 undefined나 null을 반환하면 렌더링을 하려하면 오류가 발생한다. 반면 JSX 내부에서 undefined나 null을 렌더링하는 것은 괜찮다.
  - JSX 내부에서 undefined나 null을 렌더링하면 아무것도 보여주지 않는다.
  - OR 연산자는 AND 연산자와 다르게 특정 값이 undefined나 null일 경우 보여주고 싶은 문구가 있을 때 주로 사용한다.

```
class App extends Component {
    render() {
        let name = undefined;
        return (
            <div>
                {
                    name === 'React' || <h1>This is React.</h1>
                }
            </div>
        );
    }
}
```