A. 작업환경설정

1. React 특징

1. 리액트는 `오직 View만 신경쓴 라이브러리`
2. `Virtual DOM을 사용`
   - HTML 마크업을 시각적인 형태로 변환하는 것은 DOM을 조작하는데 그 때마다 조작하는 것은 성능이 저하가 된다.
   - 리액트는 변경된 부분만 변경할 수 있도록 Virtual DOM방식을 사용하여 DOM업데이틀 추상화하여 DOM처리 횟수를 최소화
   - Virtual DOM의 처리 순서
     - 데이터를 업데이틓면 전체 UI를 Virtual DOM에 리랜더링
     - 이전 Virtual DOM에 있는 내용과 현재 내용을 비교
     - 변경 부분만 실제 DOM에 적용
3. 리액트와 Virtual DOM이 언제나 제공할 수 있는 것이 바로 `업데이트 처리의 간결성` 이다.
4. 리액트는 View만 신경쓰고 다른 라이브러리를 제공
   - `react-router(라우팅처리), axios나 fetch(AJAX처리), redux(상태관리)등을 이용`

2. 작업환경설정

1. `Node.js와 npm`
   - 리액트는 반드시 Node.js를 설치해야 한다. Node.js튼 크롬 V8자바스크립트 엔진으로 빌드한 자바스크립 런타임 모듈
   - node.js를 설치하면 `패키지 매니저 도구인 npm이 설치`
   - https://nodejs.org/ko/download 에서 LTS버전 설치후 version 확인
     cmd
     node -v
2. `yarn`
   - yarn은 npm을 대체할 수 있는 도구로 npm보다 빠르며 효율적인 태시 시스팀과 기타 부가기능을 제공
     npm install --global yarn
     yanrn --version
3. 에디터설치 : vscode
   - https://code.visualstudio.com/Download 다운 및 설치
   - 확장프로그램
     - Korean Language Pack for Visual Studio COde
     - Live Server
     - ESLint : 자바스크립 문범 및 코드 스타일 검사 도구
     - Reactjs Code Snippets (by charalampos karpidis) : 코드 스내펫모음
     - Prettier-Code formatter : 코드 스타일 자동정렬
     - CSS peek
     - Indent-rainbow

    - Matrial Icon Theme
    - Auto Close Tag
    - Auto Rename Tag
4. `git`

    - 참고사이트 : https://xangmin.tistory.com/102
5. JSX내에서 html 자동완성 설정하기

- ctrl+shift+p > settings.json
- settings.json 파일 끝에 추가

```
"emmet.syntaxProfiles": {
     "javascript": "jsx"
 },
 "emmet.includeLanguages": {
    "javascript": "html"
}
```

3. project 생성

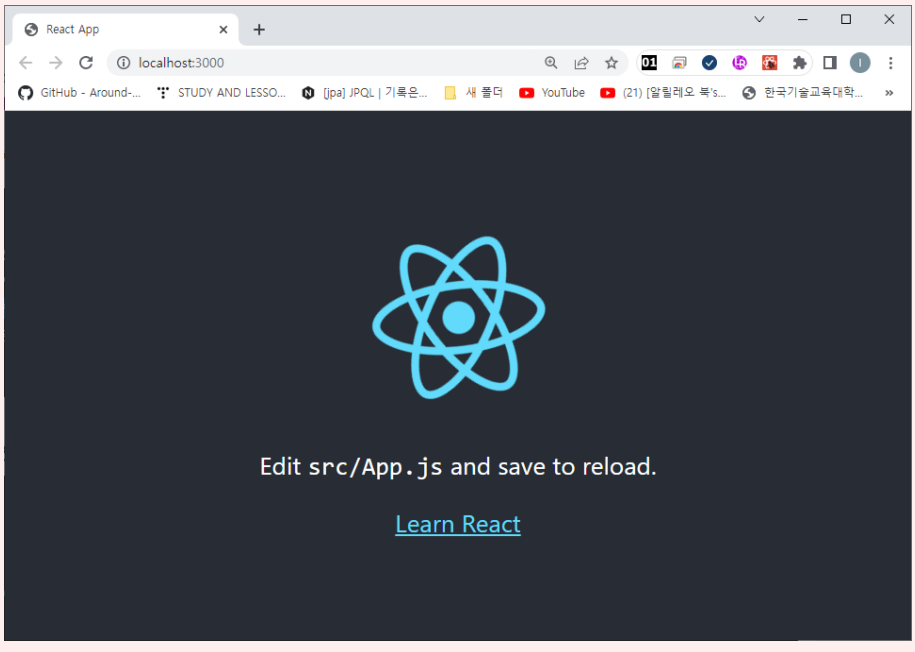
1. 프로젝트 생성 명령

- `yarn create react-app <프로젝트이름>`
- `npm init react-app <프로젝트이름>`
- `npm create-react-app <프로젝트이름>`

1. 실습용 프로젝트생성하기

    - vscode > terminal > `yarn crete react-app 01.hello-react`
2. project실행

    1. 폴더변경 : cd 01.hello-react
    2. 프로젝트시작 : `yarn start`
    3. 서버접속 : http://localhost:3000



1. package.json 구조 보기

- mindmap : https://npmgraph.js.org/

B. JSX

1. App.js

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

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>
  );
}

export default App;
```

1. `import`
    - 특정파일을 불러오는 것을 의미,
    - 브라우저가 아닌 환경에서 자바스크립트를 실행할 수 있게 해주는 node.js에서 지원하는 기능
    - node.js의 `require` 구문으로 패키지를 불러 올 수 있다.
    - 이런 기능을 이용해 브라우저에서도 `번들러 bundler` 를 사용
    - 번들러 도구를 사용하면 import(or require)로 모듈을 불러왔을 때 모듈을 합쳐서 하나의 파일을 생성
2. 상기 코드는 `App라는 컴퍼넌트를 생성`
3. 이러한 코드를 `JSX` 라고 한다. `JSX란 자바스크립트의 확장문법` 이다

```
jsx
function App() {
   return (
       <div>
           Hello <b>react</b>
       </div>
   )
}
```

```
// 상기의 코드를 아래와 같이 변환
//React.createElement(component, props, ...children)
function App() {
   return React.createElement("div", null, "Hello",
React.createElement("b", null, "react"))
}
```

2. JSX의 장점

1. 보기 쉽고 익숙하다.
2. 더 높은 활용도

3. index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

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

1. `React.StrictMod`
   - 리액트 프로젝트에서 앞으로 사라질 레거시 기능을 사용할 때 경고
   - 미래의 버전에 도입될 기능들이 정상적으호 호환될 수 있도록 `개발환경에서만 활성화 되는 디버깅용 컴퍼넌트`
2. `reportWebVitals`
   - 웹 성능을 측정하는 도구

4. JSX 문법

1. 한개의 부모 tag로 자식tag를 감싸야 한다.
2. tag를 사용하고 싶지 않을 경우 React V16부터 적용된 `<Fargment> 컴퍼넌트 기능을 사용` 하면 된다.

```
jsx
 import { Fragment } from 'react';
  function App() {
   return (
       <Fragment>
           Hello <b>react</b>
           <h1>Hello???</h1>
           <h1>React</h1>
       </Fragment>
```

```
    )
  }
```

3. Fragment는 `<> ... </>` 형태로도 사용할 수 있다.
4. JSX내부에 `{ javascript 표현식 }` 의 형태로 javascript를 사용할 수 있다.
5. `JSX내부에서는 if문을 사용할 수 없다. 대신에 조건부 연산자 즉, 삼항연산자 를 사용` 한다.

```
jsx
     function App() {
       const name = "gildong";
       return (
         <div>
           {name === 'gildong' ? (<h1>홍길동입니다!!</h1>) : (<h1>홍길동
이 아닙니다!!</h1>)}
         </div>
       );
     }
```

6. and연산자(&&)를 사용해서 조건부 랜더링을 할 수 있다.

```
jsx
   return {name === 'gildong' ? (<h1>홍길동입니다!!</h1>) : null}
   // 대신에
   return {name === 'gildong' && <h1>홍길동입니다!!</h1>}
   )
```

   - &&연산자로 조건부 랜더리을 할 수 있는 이유는 리엑트에서 false를 랜더링할 때는 null과 마찬가지로 아무것도 나타나지 않는다.
   - 주의해야할 점은 `falsy한 값인 0은 예외적으로 화면에 나타난다.`

7. undefined를 랜더링을 하면 에러가 발생한다.

```
jsx
 function App() {
   const name = undefined;

   // 1) undefined 그래로 return
   // return name;

   // 2) 대신에 or연산자를 사용
   // return name || '값이 undefiened입니다!!!';

   // 3) JSX내부에 랜더링
   return <div>{name || 'undefined값!!!'}</div>
 }
```

8. inline styling

   - DOM요소에 style을 적용할 때는 `문자열 형태가 아니라 객체형태로 정의` 해야 한다.
   - 또한 `-이 있는 경우 camel case로 작성` 해야 한다 예를 들어 background-color는 backgroundColor형태로 정의해야 한다.

```
jsx
function App() {
 const name = 'Honggildong';
 const style = {
   backgroundColor: 'gold',
   color: 'black',
```

```
    fonntSize: '48px',
    fontWeight: 'bold',
    padding: 16
  }
  return <div style={style}>{name}</div>
}
```

9. class대신에 `className`을 사용

   - css에서 class를 JPX내부에 사용할 경우는 `className`을 사용
   - class를 설정해도 적용은 되기는 하나 개발자도구 Console에서 확인하면 경고가 표시

   App.css
   ```
   .react {
    padding: 20px 20px;
    background: mediumaquamarine;
    color: whitesmoke;
    font-size: '48px';
    font-weight: 'bold';
    padding: 16;
   }
   ```
   jsx
   ```
   return <div className="react">{name}</div>
   ```
10. 꼭 닫아야 하는 tag들이 있다.

    - `<input />` self-tag or `<input></input>`처럼 사용
11. 주석

    - jsx내부에서 주석처리는 `{ /* ... */}`와 같이 작성한다.
    - `// or /*  */`는 화면에 그래로 나타난다.

5. ESLint, Prettier 적용하기

1. ESLint : View > Problem 메뉴로 확인 가능
   - terminal : `npm repo eslint`
   - 참조 : https://eslint.org/docs/latest/use/getting-started
2. Prettier

   - terminal : `npm repo prettier`
   - 옵션참조 : https://prettier.io/docs/en/options.html

   - 저장시 자동으로 코드정리하기

     - root directory(src, public등)에 `.prettierrc`작성

       ```
       {
           "singleQuote": true,
           "semi": true,
           "useTabs": false,
           "tabWidth": 2
       }
       ```

     - 메뉴 : File > Preferences > Settings : `format on save` 체크