[2022-11-15] 2반 실습 파일

학습 목표

- React의 기초와 ES6 최신 자바스크립트 문법 학습
- 리액트 JSX 및 컴포넌트 학습 및 작성

문제 풀이

1. HTML을 JSX로 변환하기

JSX이란?

자바스크립트에서 html 원소들을 편하게 작성하고 분석할 수 있게 도와주는 확장된 자바스크립트 문법입니다.

저희 브라우저 및 자바스크립트 엔진은 오래된 자바스크립트 문법으로 작성한 코드만 이해할 수 있습니다.

이렇게 확장된 문법 / 새로운 자바스크립트 문법을 사용하기 위해서는 중간에 최신말을 옛날말로 번역해주는 번역기가 필요합니다.

`번역기` === `트랜스파일러` (e.g. Babel)

+추가 용어

`컴파일러` === `인간-friendly코드 → 컴퓨터-friendly 코드 번역기`

프로그램을 실행할 때 우리가 이해하는 코드를 컴퓨터가 이해할 수 있는 0,1 binary code로 바꾸는 프로그램.

`번들러` === `엄청 많은 파일 압축기`

여러 파일들 (html, css, js, image)을 압축하는 툴. 프로젝트가 커짐에 따라 여러개의 자바스크립트 파일로 나뉘어 관리하기 시작되었고, 웹에서 다운받을 때 (배포 후 접속할 때) 파일을 여러개로 나눠받는 것보다 하나로 뭉쳐받는 것이 더 빠르기에 사용된다. (e.g. webpack. 과거에는 수동 webpack.config.js 파일 설정. 지금은 리액트 CRA(create-react-app)으로 앱 생성 시 기본 설정 완료)

▼ 문제 정답

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import * as serviceWorker from './serviceWorker';
```

```
//HTML을 JSX로 변환하여 element에 저장합니다.
// **JSX 규칙?
const element = ( // **ES6 const vs let vs var 차이?
  <>
    <h2>코더랜드에 오신것을 환영합니다:)</h2>
    <h2>즐거운 React! 함께 공부해봐요~</h2>
  </>
);

ReactDOM.render(element, document.getElementById('root'));

serviceWorker.unregister();
```

▼ 문제 확장

**JSX 규칙?

자식 태그가 여러개 있는 경우 모두 감싸주는 **상위 부모 태그**가 필요합니다. 혹은 <></> 태그로 감싸줄 수 있습니다.

소괄호로 감싸주면 코드를 그 다음 라인으로 위치시킬 수 있게하여 가독성이 좋아집니다.

**ES6 const vs let vs var 차이?

const: 변하지 않는 상수값, let & var: 변하는 값.

결론: “var 쓰지마세효…”

Why??? `호.이.스.팅`

코드가 실행되기 전 자바스크립트 엔진은 변수(로컬, 글로벌 모두)들을 모두 최상단으로 끌어올려 “아 이런 변수들이 선언되었구나”를 쭉 점검을 하고, 메모리를 할당해주는 호이스팅을 하게 됩니다.

이때 발생할 수 있는 문제점은 (1)번에서 값을 선언하기도 전에 메모리를 할당하기 때문에 Error가 출력되는 것이 아닌 undefined 값이 출력되게 됩니다.

예제2에서 또한 원래는 로컬변수로 i가 선언되었기 때문에 변수 i에 접근이 불가능해야하지만 이미 모든 변수가 메모리를 할당 받은 상태이기 때문에 (2)에서 여전히 for loop 내 로컬 변수를 외부에서 접근할 수 있게 됩니다.

```
// 예제1
console.log(a) // --- (1)
var a = 1
console.log(a)

// 예제2
for(var i = 1; i < 5; i++){
  console.log(i)
}
console.log(i) // --- (2)
```

이 문제 때문에 저희는 앞으로 let을 쓰겠습니다 (단호).

---

2. 함수에 JSX 활용하기

리액트 버전 체그: npm view react version

리액트 프로젝트 생성: create-react-app <<app_name>>

이후 최소한의 리액트 프로젝트를 구동하기 위한 파일로는 src 폴더 내 App.js & index.js,

public 폴더 내 index.html만 남기고 불필요한 주석을 제거하시면 됩니다.

Libraries?

package.json 내 dependencies를 보면 react, react-dom, react-scripts 라이브러리가 CRA시 디폴트도 설치되어있음을 알 수 있습니다. 각각의 용도는 다음과 같습니다.

- `React`: jsx 를 쓸려면 필요한 라이브러리
- `React-dom`: html요소를 화면에 불러오는 역할. 리액트 컴포넌트의 이전 state 차이를 비교 후 차이점이 있을 때 업데이트해 브라우저에 보여줌.
- `React-scripts`: 스크립트 쓸 수 있게 도와주는 도구

Scripts?

그 다음으론 scripts를 알아보겠습니다.

`start` 스크립트는 npm start 커맨드는 실행했을 때 작동하는 스크립트로 react-scripts라는 도구가 react-scripts > scripts > start.js 라는 파일을 돌리면서 프로젝트가 작동합니다. start.js 파일 내 webpack 섹션을 보면 src > index.js 파일을 엔트리로하는 소스파일을 번들링함을 알 수 있습니다.

`build` 스크립트는 npm run build를 실행했을 때 작동하는 스크립트로 같은 위치에 있는 build.js 파일을 작동시킵니다. 이는 프로젝트를 번들링 후 build라는 디렉토리에 그 결과를 저장하는 역할을 합니다. 이 build 폴더를 추후 배포할 때 사용하게되고 배포용 스크립트는 따로 작성해야합니다.

`test` 스크립트는 테스팅용 스크립트로 추후 Jest와 연결해 사용할 것입니다.

```
// index.js
import React, {useState} from 'react';
import ReactDOM from 'react-dom';
```

```
// package.json
"dependencies": {
    "@testing-library/jest-dom": "^5.16.5",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5.0",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-scripts": "5.0.1",
    "web-vitals": "^2.1.4"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
```

▼ 문제 정답

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import * as serviceWorker from './serviceWorker';

//고객의 이름을 출력하는 함수
function formatName(user) {
  return user.lastName + ' ' + user.firstName;
} //**3가지 함수 표현 방법?

//고객의 이름을 저장하는 변수
const user = {
  lastName: '코딩하는',
  firstName: '엘리스',
};

//인사문구를 출력하는 함수
//formatName()함수를 사용해 출력문구를 완성합니다.
// 자바스크립트 네이밍 컨벤션: lowerCamelCase 추가자료 https://velog.io/@cada/자바스크립트-스타일-가이드-네이밍-컨벤션-편
function getGreeting(user) {
  const formattedName = formatName(user); // **전역변수 vs 지역변수
  return <h1>Hello, {formattedName}!</h1>;
}

//getGreeting()의 결과값을 element에 저장합니다.
const element = getGreeting(user);

ReactDOM.render(element, document.getElementById('root'));

serviceWorker.unregister();
```

▼ 문제 확장

**3가지 함수 표현 방법?

```
// 1.
function formatName(user) {
  return user.lastName + ' ' + user.firstName;
}

// 2.
const formatName = (user) => {
    ...
} // 단 매개변수가 하나면 소괄호 생략 가능. 본문이 하나의 실행문인 경우도 {} 괄호 생략 가능.


// 3. 정의하자마자 바로 실행하는 함수
(function() {
function body..
})(); // onclick 이벤트 핸들러 처럼 한번만 쓰고 말 함수의 경우 사용 가능
```

**전역변수 vs 지역변수?

전역변수 (global 변수): 프로그램 전체에서 접근할 수 있는 변수. 반복적으로 사용될 변수인 경우 사용. 많이 쓰면 메모리 영향.

지역변수 (local 변수): 함수가 실행되면 만들어지고 함수가 종료되면 소멸하는 변수.

팁 💡

- 기능 단위별로 함수화를 시키기! 의존도 낮추기! 디버깅이 쉬워 유지보수성이 향상됩니다. 재사용도 용이합니다. 가독성에 좋습니다.

---

3. 함수 컴포넌트 만들어보기

컴포넌트란?

부품입니다. 벽돌을 구울 때마다 틀을 계속 만들면 힘듭니다. 벽돌 굽는 툴을 하나 만들어놓고 계속 반복해서 사용합니다. 반복적인 코드가 줄어들고 + 유지보수 & 디버깅이 쉬워지며 + 협업하기도 아주 굿굿 + 확장도 수월합니다. (eg. 자식 컴포넌트를 부모 컴포넌트에 전달하고 싶은 경우) + 관리가 수월합니다. 다른 웹개발 같은 경우는 뷰, 컨트롤러, 라우터 나눠서 관리하는데 컴포넌트를 사용하면 한번에 관리가 가능합니다.

리액트 디렉토리 구조?

제가 주로 리액트로 개발할 때 자주 쓰는 디렉터리 구조로 components, routes 폴더를 만들어 그 안에 다음과 같은 컴포넌트 파일을 저장합니다.

> routes 폴더 > 특정 라우트로 접속했을 때 전달할 페이지 컴포넌트 파일 위치
>
> components 폴더 > 페이지 컴포넌트를 구성하는 작은 단위의 컴포넌트 사용

뿐만 아닌 일반 웹사이트의 경우에도 components > partials > footer.js, header.js 처럼 컴포넌트를 만들어 놓고 베이스가 되는 base.js를 만들어 상속하면 모든 페이지에서 쓸 수 있습니다.

▼ 문제 답안

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';

const name = 'Sara'; // 전역변수 - 또 사용할 것 같으면 선언
function Welcome() { // 컴포넌트는 대문자
  return <h2>Welcome, {name}!</h2>;
} //**과거 클래스 컴포넌트 문법?

const element = <Welcome />;

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

▼ 문제 확장

**과거 클래스 컴포넌트 문법?

```
class Profile extends React.Component { // step 1. 확장
  constructor() { // 새로운 변수 만드는 공간
    super(); // 2. super 밑에 state 만들기
    this.state = { name: 'Sara' }
  }

//  changeName = () => {
//    this.setState( { name: "hahahaa"}); // 3. state 값 바꾸기
//  }

// 4. 꺼내 쓸 때는 this.state.state명
  render() {
    return (
      <>
        <h2>Welcome, {this.state.name}!</h2>
        //<button onClick={ () => {this.setState( {name: 'Park'} )}}> 버튼 </button>
        //<button onClick={ this.changeName } > 버튼 </button>
      </>
    );
```

```
  }
}
```

---

4. 컴포넌트 합치기

▼ 문제 답안

```
// index.js

import React from 'react';
// Profile 함수 import
import { Profile } from './components/Profile.js'; // **import?

function App() {
  const title = '사용자 프로파일';
  // Profile 함수 컴포넌트 삽입
  return (
    <div>
      <h1>{title}</h1>
      <Profile />
    </div>
  );
}

export default App;
```

```
// Profile.js

import React from 'react';

// Comment 컴포넌트 import
import { Comment } from './Comment.js';

// UserInfo 컴포넌트 import
import { UserInfo } from './UserInfo.js';

import '../index.css';

function Profile() {
  const user1 = {
    name: '엘리스 토끼',
    age: '12',
  }; // 데이터 꺼낼 때 users1.name
  const text1 = 'React는 재밌다!!';

  // Comment 컴포넌트를 삽입하며, text props로 text1 를 전달함.
  // 컴포넌트를 삽입하며, user props로 user1 를 전달함.

  return (
```

```
    <div className="profile"> // props는 lowerCamelCase로
      <Comment text={text1} />
      <UserInfo user={user1} />
    </div>
  );
}

export { Profile };
```

▼ 문제 확장

**import?

import {<<컴포넌트명>>} vs import <<컴포넌트명>>?

export하는 파일 내에서 export default 로 export를 하면 import 할 때 {} 괄호 없이 바로 불러올 수 있습니다.

그게 아닌 export만을 사용해 export한 경우 괄호가 필요합니다.

export default는 파일당 단 한번만 사용 가능하기 때문에 export할 함수가 하나인 경우, 혹은 여러 개여도 그 대표 함수로 여기는 함수에 한해 사용합니다. import 할 때 이름 변경이 가능합니다. (eg. book으로 이름 변경. import {React as book} from “react”;

현재 파일의 위치를 파악할 때 다음 커맨드를 사용할 수 있습니다.

> console.log('__dirname', __dirname); // 파일명 미포함 절대 경로
> console.log('__filename', __filename); // 파일명 포함 절대 경로

현재 위치에서 상대적인 위치로 이동한다고 했을 때 다음 심볼을 사용합니다.

> ./ > 현 위치와 같은 레벨에 있는 파일/폴더로 이동
> ../ > 현 위치보다 상위 레벨로 이동 (바깥)

/ > root 디렉토리로 이동 (~~최최상단 프로젝트 폴더~~ 가장 최상의 디렉토리)

▼ ***(추가) 절대 경로를 사용하여 모듈 임포트 하기***

1. ***baseurl 설정***

설정 전 임포트 방법 === (1). 후 임포트 방법 === (2).

```
// jsconfig.json
{
    "compilerOptions": {
      "baseUrl": "src" // src 폴더를 기준으로 절대 경로를 사용하겠다
      // 기본경로가 여기서 시작합니다
    },
    "include": ["src"]
}
```

```
// Profile.js

import Footer from "components/Footer" // ---(2)
import Footer from "../../components/partials/Footer"; // ---(1)

const Profile = () => {
    return <Footer/>
}
export default Profile;
```

2. ***자주 임포트를 사용하는 프로젝트 path에 별명을 붙여 사용해보자***

아래 사진과 같은 프로젝트 구조에서 Profile.js파일에서 Footer.js 파일을 import 해주기 위해서는 이 임포트 라인이 필요합니다.

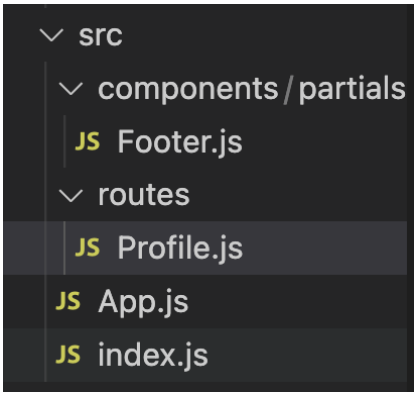
> import Footer from "../../components/partials/Footer";

컴포넌트 디렉토리 개수가 많아지거나 서로 멀리 떨어진 컴포넌트를 참조하게 되면 경로가 많이 지저분해질 수 있습니다.

저희는 힘들 걸 싫어하기 때문에 절대경로를 생성해주면 그 수고를 덜어줄 수 있습니다.

임포트 라인이 있는 파일은 절대 경로를 참조하기에 파일 위치를 변경해서 문제 없습니다.

먼저 프로젝트 최상위 폴더(root 프로젝트 위치)에 jsconfig.json 파일 추가해줍니다.



```
// jsconfig.json

{
    "compilerOptions": {
      "baseUrl": "src",// 기본경로가 여기서 시작합니다
      "paths": {
        "@components/*": ["./components/*"], // 컴포넌트 디렉토리 절대 경로 설정
        "@pages/*": ["./pages/*"]
      }
    }
```

```
// Profile.js

import Footer from "@/components/partials/Footer"; // 절대 경로 불러오는 방법
// import Footer from "../../components/partials/Footer";

const Profile = () => {
    return <Footer/>
}
export default Profile;
```

위와 같은 방식으로 절대 경로에 별명을 붙여주게 되는데….

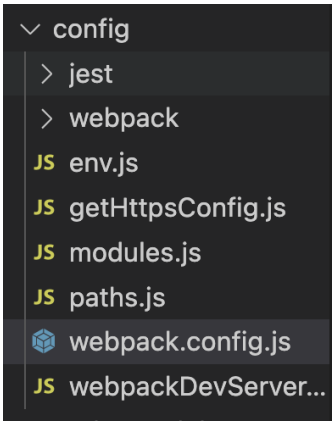
하지만 이렇게 별명을 붙여주기 위해서는 CRA로 생성한 프로젝트의 webpack 설정을 수정해주어야합니다…당장은 아니더라도 언젠간 쓰실테니 정리해서 올려놓겠습니다…

Step 1. npm run eject 커맨드 실행 (수업때는 안다루고 넘어갔지만 바로바로 숨겨진 모든 설정을 밖으로 추출해주는 명령어입니다)

(*Remove untracked files, stash or commit any changes, and try again.* 에러 발생시 깃커밋해줍니다.)

Step 2. 성공적으로 커맨드가 실행되면 config라는 폴더가 생성되고 그 안 webpack.config.js내 설정을 변경해주겠습니다.

ctrl+F로 (세미콜론을 포함한) “`resolve:`” 키워드를 검색해주세요. 하단의 코드박스 내 웹팩이 이해할 수 있도록 앞서 추가한 @별명을 추가해주세요.




```
resolve: {
  // ...
  alias: {
  // ...
  '@components': path.resolve(__dirname, '../src/components/*'),
  '@pages': path.resolve(__dirname, '../src/pages/*') // 웹팩에게 우리 별명 알려주기
  },
}
```

완료! 소스코드는 깃랩 set_absolute_path_w_react 디렉토리에서 확인 고!
