[스파르타코딩클럽] 리액트 심화반 1주차

[수업 목표]

1. React 프로젝트를 세팅할 수 있다.
2. React 기본기를 익힌다.
3. 자바스크립트 문법을 정리한다.
4. 함수형 컴포넌트와 훅을 다룰 수 있다.

[목차]

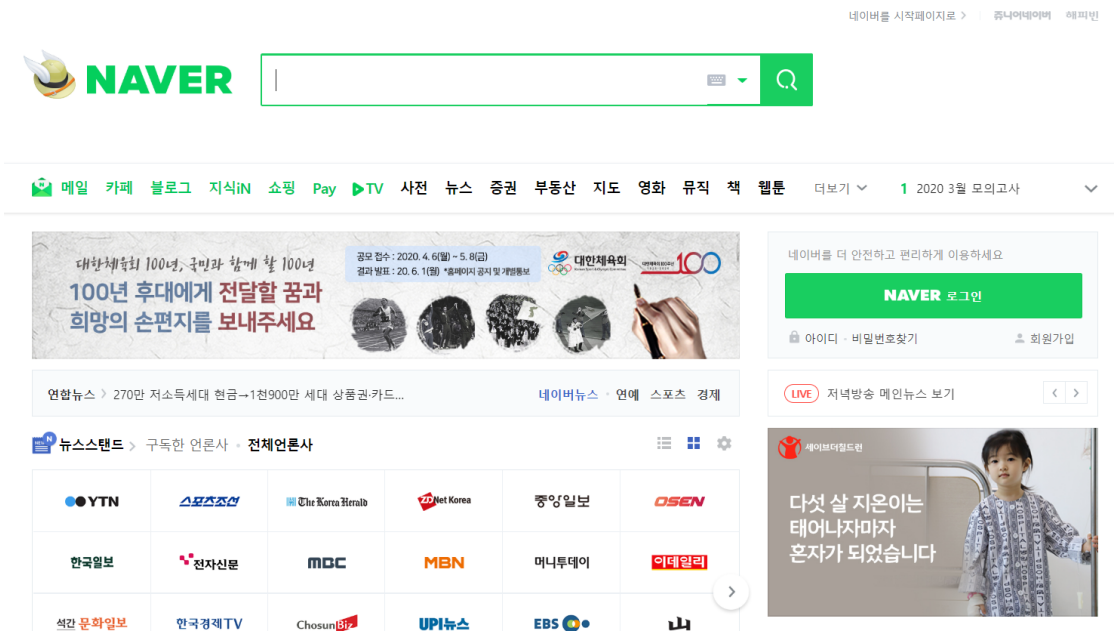
---

01. 오늘 배울 것

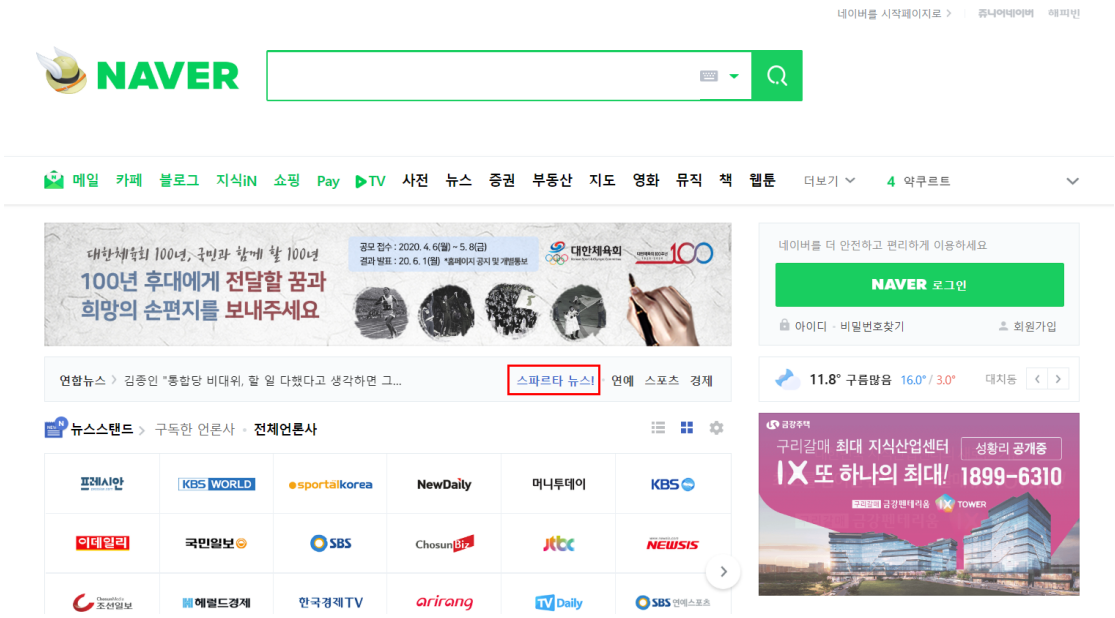
▼ 1) 프로젝트 세팅하는 법, 클라이언트 환경, 리액트와 자바스크립트 기본기

▼ 1. 클라이언트 환경을 알아보자

> 👉 크롬 창에서 보이는 웹페이지는 어떤 원리로 보여지는 것일까요?
> 원리를 알기 위해서, 일단 해킹부터 해보죠! (응?)
>
> 함께 해보아요 → '네이버뉴스'라고 쓰인 곳을 원하는 문구로 바꾸어보죠!



👉 앗! 바뀌었다!

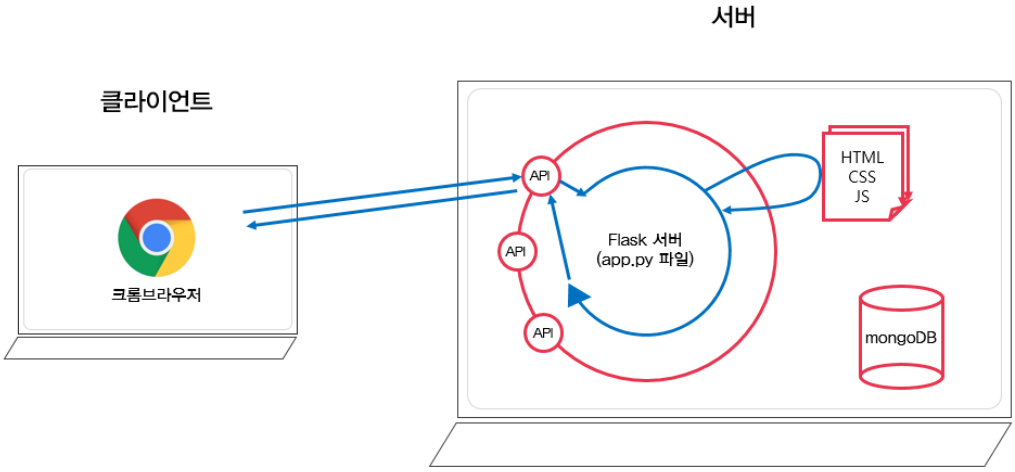


▼ 2. 리액트와 자바스크립트 기본기

👉 네! 우리가 보는 웹페이지는 모두 서버에서 미리 준비해두었던 것을 "받아서", "그려주는" 것입니다. 즉, 브라우저가 하는 일은 1) 요청을 보내고, 2) 받은 HTML 파일을 그려주는 일 뿐이죠.

👉 근데, 1)은 어디에 요청을 보내냐구요? 좋은 질문입니다. 서버가 만들어 놓은 "API"라는 창구에 미리 정해진 약속대로 요청을 보내는 것이랍니다.

예) https://naver.com/
→ 이것은 "naver.com"이라는 이름의 서버에 있는, "/" 창구에 요청을 보낸 것!



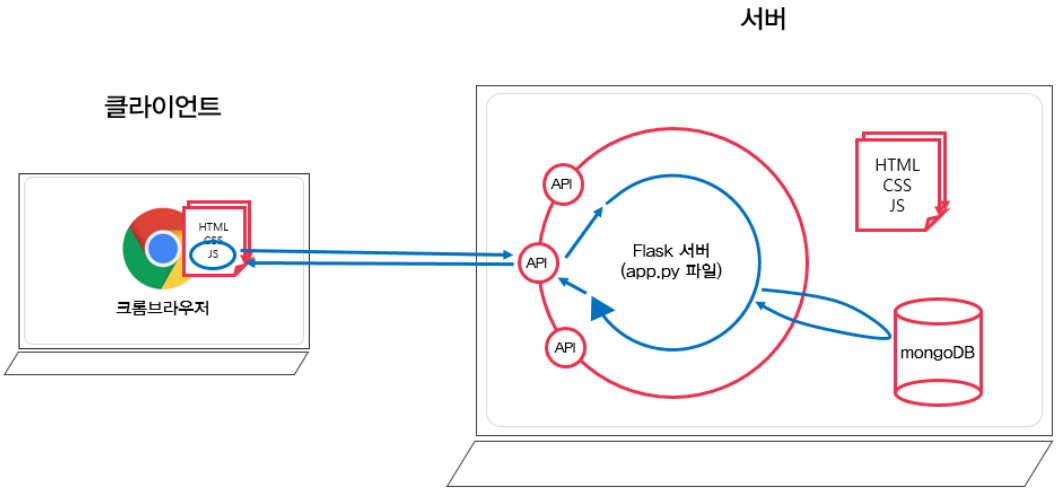
▼ 3. 이벤트

👉 앗, 그럼 항상 이렇게 HTML만 내려주냐구요?
아뇨! 데이터만 내려 줄 때가 더~ 많아요.

사실 HTML도 줄글로 쓰면 이게 다 '데이터'아닌가요?

👉 자, 공연 티켓을 예매하고 있는 상황을 상상해봅시다!
좌석이 차고 꺼질때마다 보던 페이지가 리프레시 되면 난감하겠죠ㅠ?

이럴 때! 데이터만 받아서 받아 끼우게 된답니다.



👉 데이터만 내려올 경우는, 이렇게 생겼어요!
(소곤소곤) 이런 생김새를 JSON 형식이라고 한답니다.

openapi.seoul.go.kr:8088/6d4d7 ×

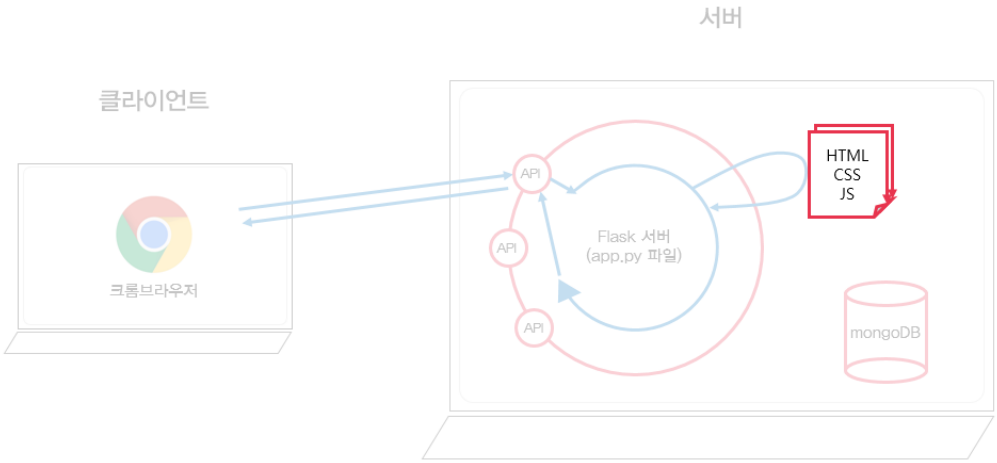
주의 요함 | openapi.seoul.go.kr:8088/6d4d776b466c656533356a4b4b5872/json/RealtimeCityAir/1/99


```
{
  - RealtimeCityAir: {
        list_total_count: 25,
      - RESULT: {
            CODE: "INFO-000",
            MESSAGE: "정상 처리되었습니다"
        },
      - row: [
          - {
                MSRDT: "202004241900",
                MSRRGN_NM: "도심권",
                MSRSTE_NM: "중구",
                PM10: 44,
                PM25: 20,
                O3: 0.039,
                NO2: 0.02,
                CO: 0.4,
                SO2: 0.003,
                IDEX_NM: "보통",
                IDEX_MVL: 59,
                ARPLT_MAIN: "PM10"
            },
          - {
                MSRDT: "202004241900",
```


▼ 4. 1~5주차에 배울 순서!

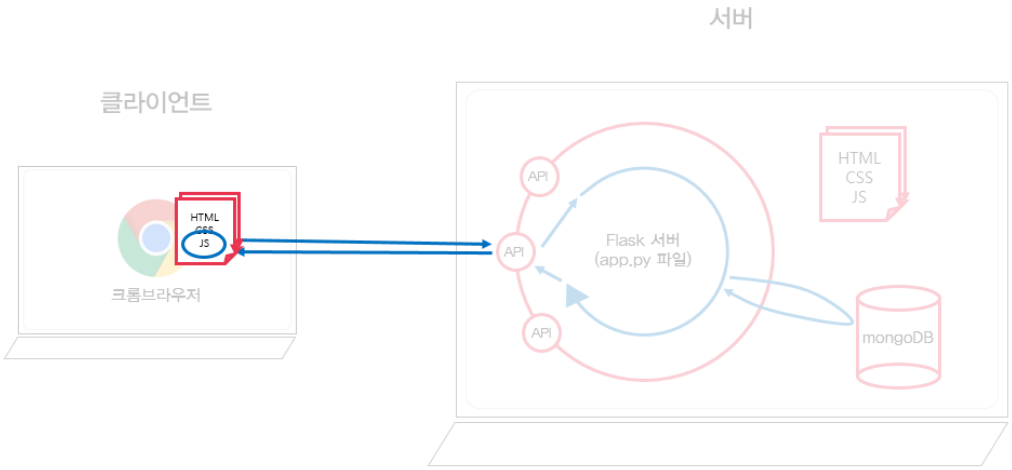
▼ 1주차: HTML, CSS, Javascript

👉 오늘은 HTML과 CSS를 배우는 날! 즉, 4주차에 내려줄 HTML파일을 미리 만들어 두는 과정입니다. + 또, 2주차에 자바스크립트를 능숙하게 다루기 위해서, 오늘 문법을 먼저 조금 배워둘게요!



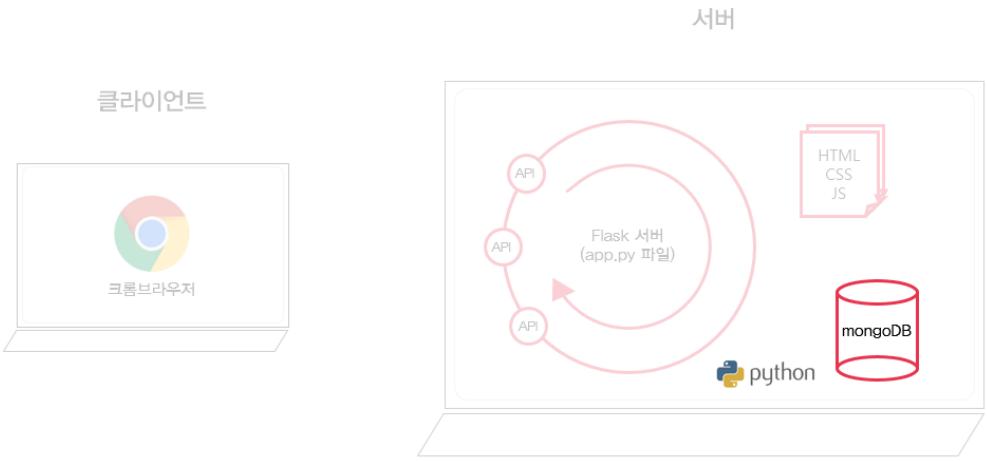
▼ 2주차: jQuery, Ajax, API

> 👉 오늘은 HTML파일을 받았다고 가정하고, Javascript로 서버에 데이터를 요청하고 받는 방법을 배워볼거예요



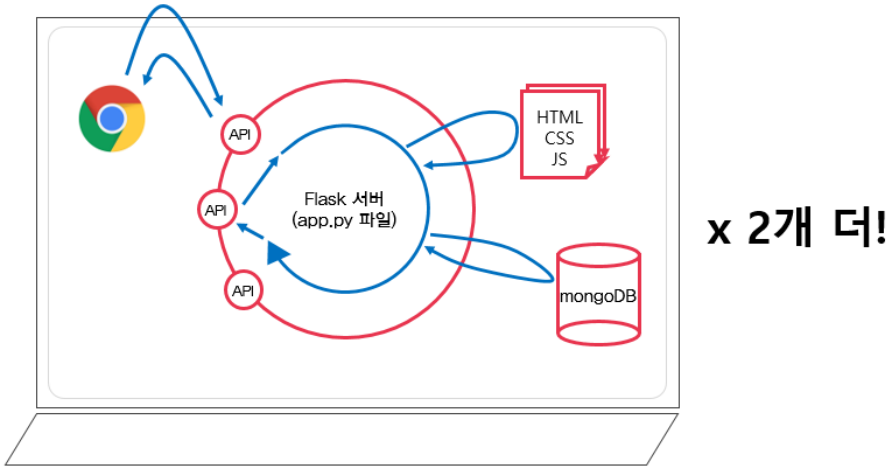
▼ 3주차: Python, 크롤링, mongoDB

> 👉 오늘은 드디어 '파이썬'을 배울거예요. 먼저 문법을 연습하고, 라이브러리를 활용하여 네이버 영화목록을 쫙 가져와보겠습니다. (기대되죠!)
> +
> 그리고, 우리의 인생 첫 데이터베이스. mongoDB를 다뤄볼게요!



▼ 4주차: 미니프로젝트1, 미니프로젝트2

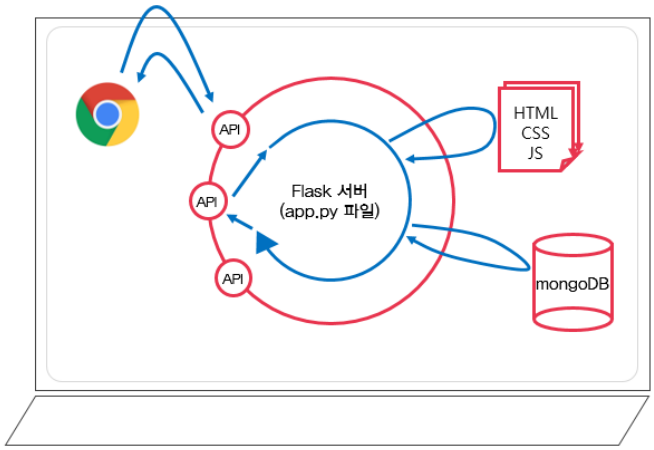
👉 오늘은 서버를 만들어봅니다! HTML과 mongoDB까지 연동하여,
미니프로젝트1, 2를 완성해보죠! 굉장히 재미있을 거예요!



x 2개 더!

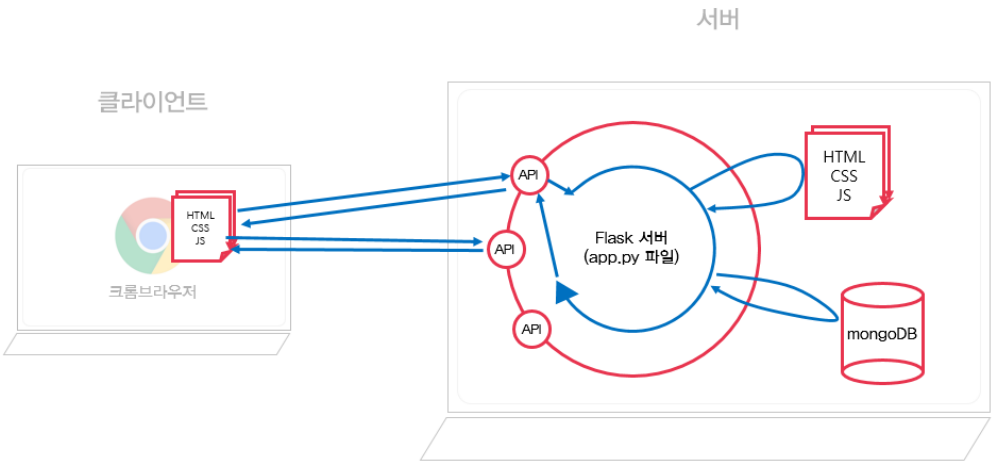
👉 나중에 또 이야기하겠지만 헷갈리면 안되는 것!
우리는 컴퓨터가 한 대 잖아요... 그래서 같은 컴퓨터에다 서버도 만들고, 요청도 할 거예요. 즉, 클라이언트 = 서버가 되는 것이죠.

이것을 바로 "로컬 개발환경"이라고 한답니다! 그림으로 보면, 대략 이렇습니다.



▼ 5주차: 미니프로젝트3, AWS

> 👉 오늘은 아직 익숙해지지 않았을 당신을 위해! 같은 난이도의 유사한 한 개의 프로젝트를 더 진행합니다.
>
> 그치만 우리 컴퓨터를 24시간 돌려둘수는 없잖아요!
>
> 그래서 두구두구.. 인생 첫 배포!를 해볼 예정입니다! 클라우드 환경에서 컴퓨터를 사고, 거기에 파일을 올려 실행해보겠습니다.



02. Javascript Re-start! (1) - 기본

> 💡 꼭 알아야하는 자바스크립트 기본 문법을 짧게 요약했어요. 😉

▼ 3) 변수와 상수

> 👉 **[변수 생성의 3단계]**
> 선언 → 초기화 → 할당
> - 선언: 실행 컨텍스트에 변수 객체를 등록 (스코프가 참조하는 대상이 되도록요!)
> - 초기화: 변수 객체에 등록된 변수를 위해 메모리에 공간을 확보 (여기서 변수는 보통 undefined 로 초기화됩니다!)
> - 할당: undefined로 초기화된 변수에 실제 값을 할당

▼ i) var

> 👉 var는 가급적 사용하지 않는 게 좋아요!

- var로 선언한 변수는 블록 스코프가 아니라 함수 수준 스코프를 가진다.
- var는 선언과 초기화를 한번에 한다!
- 재선언이 가능하다.
- 선언하기 전에도 사용할 수 있다.

```
// var는 이런 식의 사용도 가능합니다. 괴상하죠!
// var name은 선언! name = "perl"은 할당!
function cat(){
  name = "perl";
  alert(name);
  var name;
}

cat();
```

- 코드 블럭을 무시합니다. (var는 함수의 최상위로 호이스팅 됩니다. 선언은 호이스팅 되고 할당은 호이스팅 되지 않습니다.)

```
// var name은 함수의 최상위로 호이스팅되기 때문에, 실행될 일 없는 구문 속에 있어도 선언이 됩니다.(자바스크립트가 동작하기 전에 코드를 한 번 훑는데 그 때, var로 선언된 코드를 전부 최상위로 끌어올려버립니다. 동작해선 안되는 구문이라도... 몽땅..!)

function cat(){
  name = "perl";
  // 이 if문 내로는 절대 들어올 일이 없죠!
  if(false) {
    var name;
  }
  alert(name);
}

cat();
```

▼ ii) let

- 자바스크립트에서 변수를 생성할 때 쓰는 키워드
- block-scope를 갖는다.

    - `{}`안에서 선언하면 `{}` 안에서만 쓰고 바깥에선 쓸 수 없어요!
- 재선언은 불가, 재할당은 가능!

```
// 재할당은 가능!
let cat_name = 'perl';
cat_name = '펄이';

// 재선언은 오류!
let cat_name = 'perl';
let cat_name = '펄이';
```

▼ iii) const

- 자바스크립트에서 상수를 생성할 때 쓰는 키워드
- block-scope를 갖는다.
    - `{}`안에서 선언하면 `{}` 안에서만 쓰고 바깥에선 쓸 수 없어요!
- 재선언 불가, 재할당도 불가! (⇒ 선언과 동시에 할당해요!)

```
// 재할당 오류!
const cat_name = 'perl';
cat_name = '펄이';

// 재선언도 오류!
const cat_name = 'perl';
const cat_name = '펄이';

// 선언과 동시에 할당 되기 때문에 값을 안줘도 오류가 납니다.
// declare!
const cat_name;
```

▼ iv) TDZ(Temporal Dead Zone) = 일시적 사각지대

▼ 1) let과 const도 호이스팅이 될까?

👉 **[간단하게 말하자면...]**
var과 let, const의 차이점 중 하나는 변수가 선언되기 전에 호출하면 `ReferenceError`가 난다는 점이에요!

Q. 왜 에러가 날까? 호이스팅이 안 된 걸까?
A. **호이스팅(=선언 끌어 올리기)은 됩니다!** 다만, 선언한 후, **초기화 단계에서 메모리에 공간을 확보**하는데, 선언을 호이스팅해도 초기화 전까지 메모리에 공간이 없죠! 그래서 변수를 참조할 수 없기 때문입니다.
이걸 TDZ라고 해요!

- 앗, 그럼 let, const는 호이스팅이 되지 않는 건가?

  → 답은 **호이스팅이 된다!**
- 그럼 왜 레퍼런스 에러가 날까?
  → TDZ 때문에 그렇습니다!
- [3줄 요약]

→ let, const 선언도 호이스팅 된다.

→ 스코프에 진입할 때 변수를 만들고, TDZ가 생성되지만 코드 실행이(=실행 컨텍스트가) 변수가 있는 실제 위치에 도달할 때까지 엑세스를 못할 뿐!

→ 자주 물어보는 기초예요!

> 💡 **[알면 재미있는 이야기: 변수명]**
> 변수명은 숫자로 시작할 수 없고, _와 $를 제외한 특수문자를 쓸 수 없어요!
> 그 외의 모든 것은 가능합니다. 심지어, 한글도 가능해요. 🤭

▼ 4) 자료형

> 👉 자바스크립트는 8가지 기본 자료형을 지원합니다!
> **객체를 제외한 나머지 7가지를 원시형(primitive type)**이라고 불러요.
> `typeof` 연산자로 자료형을 알아낼 수 있어요. 🙂

- 정수, 부동 소수점을 저장하는 `숫자형`: -(2^53-1) ~ (2^53-1)까지 지원
- 아주 큰 숫자를 저장하는 `BigInt형`
- 문자열을 저장하는 `문자형`
- 논리 값 (true/false. `boolean형`)
- 값이 할당되지 않음을 나타내는 독립 자료형 `undefined`
- 값이 존재하지 않음을 나타내는 독립 자료형 `null`
- 복잡한 자료구조를 저장하는 데 쓰는 `객체형`(다음 강의에서 조금 더 알아봅니다!)
- 고유 식별자를 만들 때 쓰는 `심볼형`

> 💡 **[알아두면 더 좋은 Tip: 엄격 모드]**
> 자바스크립트는 엄청 많은 버전이 있습니다! 우리가 들어본 ES6도 사실 자바스크립트의 한 버전일 뿐입니다. (보통 이야기하는 하위 버전 자바스크립트는 ES3, 최신이다! 모던이다! 하면 ES6 이상을 가리키죠!)
>
> 자바스크립트는 하위 호환성에 도른 언어라, 코드를 읽을 때 모던한 방식보다는 하위 호환에 초점을 맞춰 동작합니다. 최신 기능이라기에 써봤는데, 오류를 뿜어낼 수 있단 소리입니다. 😢
>
> 그럴 때 쓰는 게 엄격 모드 `use strict`입니다. `'use strict;'`를 스크립트 최상단에 써주면 모던 자바스크립트에서 지원하는 모든 기능을 활성화해줍니다!

03. Javascript Re-start! (2) - 객체

▼ 5) 객체란?

- 오직 한 타입의 데이터만 담을 수 있는 원시형과 달리, 다양한 데이터를 담을 수 있다.

- key로 구분된 데이터 집합, 복잡한 개체를 저장할 수 있다.
- `{...}` ← 중괄호 안에 여러 쌍의 프로퍼티를 넣을 수 있다.
  - 프로퍼티는 `key : value`로 구성
  - key에는 문자형, value에는 모든 자료형이 들어갈 수 있음

```
// 객체 생성자로 만들기
let cat = new Object();

// 객체 리터럴로 만들기
// 중괄호로 객체를 선언하는 걸 리터럴이라고 하는데, 객체 선언할 때 주로 씁니다!
let cat = {};
```

▼ 6) 상수는 재할당이 안 된다고 했지만...

- const로 선언된 객체는 수정될 수 있어요. 😖 (← 아찔한 여러분,,)
- const로 선언된 객체는 객체에 대한 참조를 변경하지 못한다는 것을 의미합니다!
- 즉, 객체의 프로퍼티는 보호되지 않아요!

```
// my_cat이라는 상수를 만들었어요!
const my_cat = {
  name: "perl",
  status: "좀 언짢음",
}

my_cat.name = "펄이";

console.log(my_cat) // 고양이 이름이 바뀌었죠!

// 여기에선 에러가 날거예요. 프로퍼티는 변경이 되지만, 객체 자체를 제할당할 순 없거든요!
my_cat = {name: "perl2", status: "많이 언짢음"};
```

04. Javascript Re-start! (3) - 함수

💡 자바스크립트는 함수를 **특별한 값** 취급을 해요.
자바스크립트는 ()가 있으면 함수를 실행하고 ()가 없으면 함수를 문자형으로 바꿔 출력하기도 서슴치 않거든요. (함수를 값으로 취급하는거죠!)
이걸 응용하면, 함수를 복사할 수 있고, 또 매개변수처럼 전달할 수 있어요. (소근)

- 함수는 기본적으로 undefined를 반환해요. return으로 어떤 값을 넘겨주지 않는다면요!

▼ 7) 함수 선언문과 함수 표현식

- 함수 선언문

```
// 이렇게 생긴 게 함수 선언문 방식으로 함수를 만든 거예요.
function cat() {
```

```
  console.log('perl');
}
```

- 함수 표현식

```
// 이렇게 생긴 게 함수 표현식을 사용해 함수를 만든 거예요.
let cat = function() {
  console.log('perl');
}

// 물논 화살표 함수로 써도 됩니다.
// 다만 주의하실 점! 화살표 함수는 함수 표현식의 단축형이라는 거! 주의하세요! :)
let cat2 = () => {
  console.log('perl2');
}
```

- 함수 선언문 vs 함수 표현식
  - 함수 선언문으로 함수를 생성하면 독립된 구문으로 존재
  - 함수 표현식으로 함수를 생성하면 함수가 표현식의 일부로 존재
  - 함수 선언문은 코드 블록이 실행되기 전에 미리 처리되어 블록 내 어디서든 사용할 수 있다.
  - 함수 표현식은 실행 컨텍스트가 표현식에 닿으면 만들어진다. (변수처럼 처리되는 거죠!)

▼ 8) 지역 변수와 외부 변수

- 지역 변수
  - 함수 내에서 선언한 변수
  - 함수 내에서만 접근 가능
- 외부 변수(global 변수라고도 합니다.)
  - 함수 외부에서 선언한 변수
  - 함수 내에서도 접근할 수 있다.
  - 함수 내부에 같은 이름을 가진 지역 변수가 있으면 사용할 수 없다.

```
let a = 'a';
let b = 'b';
let c = 'outter!';
const abc = () => {
  let b = 'inner!';
  c = 'c';
  let d = 'd';
  console.log(a, b, c, d);
}

console.log(a, b, c, d); // a, b, outter, undefined

abc(); // a, inner, c, d

console.log(a, b, c, d); // a, b, c, undefined
```

▼ 9) 콜백 함수

👉 함수를 값처럼 전달할 때, 인수로 넘겨주는 함수를 콜백 함수라고 해요!

- 예시

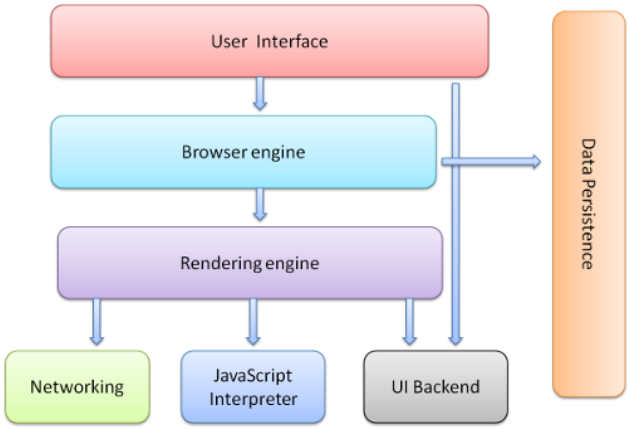
```
const playWithCat = (cat, action) => {
  action(cat);
}

const useBall = (cat) => {
  alert(cat+"과 공으로 놀아줍니다.");
}

//playWithCat 함수에 넘겨주는 useBall 함수가 콜백 함수입니다!
playWithCat("perl", useBall);
```

05. 브라우저 톺아보기

▼ 10) 브라우저란?

⚠️ 브라우저는 엄청 많은 기능을 가진 고도화된 어플리케이션입니다.
자바스크립트는 바로 이 브라우저에서 사용하기 위해 만들어졌어요. 😊
오늘은 브라우저를 이루고 있는 것들과 우리에게 제일 중요한 **브라우저가 어떻게 뷰를 화면에 그리는 지**를 알아봅시다.

- 브라우저 구성



  - 사용자 인터페이스 : 주소 표시줄, 이전/다음 페이지 버튼, 북마크, 새로고침 버튼 등
  - 브라우저 엔진 : 사용자 인터페이스와 렌더링 엔진 사이의 동작을 제어
  - 렌더링 엔진 : 화면을 창에 보여줌
  - 통신 : HTTP같은 프로토콜을 이용해서 서버에 요청을 보내는 등에 사용
  - UI 백엔드 : 얼럿 창, 셀렉트 박스, 콤보 박스 등을 OS 별로 그림
  - 자바스크립트 해석기 : 자바스크립트를 해석하고 실행

- 자료 저장소 : Cookie, Local Storage, Session Storage 등 데이터 저장

> ⚠️ 브라우저 구성에 관해 더 자세한 내용을 살펴보고 싶다면 아래 글을 참고하세요!
> 브라우저는 어떻게 동작하는가? → (링크)

▼ 11) 렌더링 엔진 동작

> ⚠️ 브라우저가 화면을 그리는 과정을 알아봅시다!

- 파싱 단계 : DOM 트리와 CSSOM 트리를 만듭니다.
- 렌더 트리 단계 : DOM과 CSSOM을 묶어서 렌더 트리를 만듭니다.
- 레이아웃 단계 : 렌더 트리에서 각 노드가 어디에 어떻게 그려져야 하는 지 모양을 계산합니다.
- 페인트 단계 : 계산한대로 화면에 엘리먼트들을 배치합니다. (페인팅한다고 해요!)

▼ 12) DOM이란?

> ⚠️ DOM(Document Object Model)
> MDN 문서에서는 아래와 같이 설명합니다.
>
> [문서 객체 모델(The Document Object Model, 이하 DOM) 은 HTML, XML 문서의 프로그래밍 interface 이다. DOM은 문서의 구조화된 표현(structured representation)을 제공하며 프로그래밍 언어가 DOM 구조에 접근할 수 있는 방법을 제공하여 그들이 문서 구조, 스타일, 내용 등을 변경할 수 있게 돕는다. DOM 은 nodes와 objects로 문서를 표현한다. 이들은 웹 페이지를 스크립트 또는 프로그래밍 언어들에서 사용될 수 있게 연결시켜주는 역할을 담당한다.]
>
> MDN에서 설명하는 내용이 이해하기 어렵다면 딱 이렇게만 기억하세요.
> **웹페이지는 document고, document는 객체예요.**
> DOM은 웹페이지의 모든 콘텐츠를 객체로 나타냅니다. 그리고 이 객체는 수정할 수 있어요.
> document 객체를 사용해 웹 페이지 내의 무엇이든 변경할 수 있다는 이야기예요.
>
> 이게 전부예요! 😊
>
> MDN에서 더 알아보기→ (링크)
> 명세서 보기→ (링크)

▼ 13) BOM

⚠️ BOM(Browser Object Model)
navigator, location 같은 객체를 보신 적 있나요? 이런 객체들은 브라우저가 제공하는 추가 객체입니다. 네! 호스트 환경을 기억하고 있죠? 자바스크립트가 돌아가는 플랫폼이요.
BOM은 호스트인 브라우저가 제공하는 추가 객체를 말합니다.
navigator, location외에도 confirm, alert 등이 BOM의 일부예요.
confirm, alert 등은 사용자와 브라우저 사이의 원활한 소통을 도와주는 순수한 브라우저 메서드거든요.

▼ 14) CSSOM

⚠️ CSSOM(CSS Object Model)
CSS는 HTML하고는 구조가 다르죠?
CSS만의 문법, 규칙이 있잖아요. 그 규칙과 스타일 시트를 객체로 나타냅니다. DOM이 웹 페이지를 객체로 나타내는 것처럼요.

명세서 보기 → (링크)

06. CRA로 첫 리액트 프로젝트 세팅하기

▼ 15) 첫 리액트 프로젝트 만들기

👉 CRA는 리액트 기초 과정에서 이미 여러번 다뤄보았죠!
빠르게 환경설정하고 CRA로 프로젝트를 하나 만들어봅시다.

▼ (1) nvm으로 node 설정하기

```
nvm -v # nvm이 잘 설치되었나 보기 위해, 버전 정보를 확인해봅니다.
nvm install 16.14.0 # 현재 LTS 버전으로 설치해봅시다!
nvm use 16.14.0 # 설치한 노드를 사용해봅시다.
```

▼ (2) node로 yarn 설치하기

```
node -v # node 버전을 확인해봅니다.
npm -v # npm은 노드의 패키지 매니저예요. 이걸 통해서 누군가 만들어 둔 패키지를 설치하고 사용할 수 있어요!
npm install -g yarn # -g는 글로벌로 설치하겠다!고 알려주는 옵션입니다. yarn을 글로벌로 설치해봅시다!
```

▼ (3) yarn으로 create-react-app 설치하기

```
yarn -v # yarn 버전을 확인해봅니다.
# global은 글로벌 설치 옵션입니다.
# yarn은 npm과 명령어가 조금 다른데요, install 대신 add를 씁니다.
# 이런 명령어의 차이는 공식문서를 통해 확인해도 좋고, yarn --help로 확인해도 좋습니다.
```

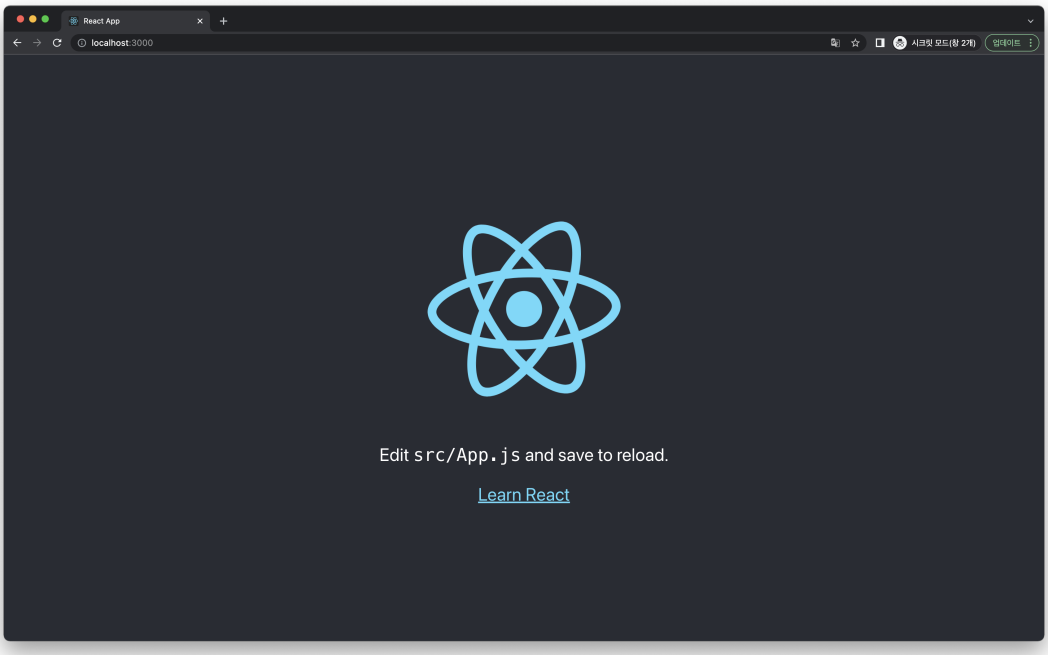
```
# yarn으로 create-react-app을 설치합니다.
yarn add create-react-app global
```

▼ (4) create-react-app으로 프로젝트 만들기

```
yarn create react-app firstprj # 첫번째 프로젝트를 만들어봅시다!
```

▼ (5) 브라우저로 확인해보기

> 👉 프로젝트를 다 만들었다면 브라우저에서 확인도 해봐야죠!
> 브라우저를 열고 localhost:3000으로 이동해서 확인해봅시다.
> 아래와 같은 화면이 나왔다면 첫번째 프로젝트를 잘 만든거예요. 😊



07. CRA 없이 리액트 프로젝트 세팅하기

▼ 16) webpack

> 👉 번들러(Bundler)라는 단어를 들어본 적 있나요?
> 아마 심화반을 듣는 분들은 한 번 쯤 들어보셨을거예요. 번들러는 여러 개로 나뉜 모듈을 하나로 묶어주는 거예요. 의존성이 있는 것들을 찾아서 그룹핑을 해주는 도구죠!
> 이번 시간에 배울 웹팩은 엄청 많이 쓰이는 오픈 소스 **모듈 번들러 라이브러리**입니다. 😊

▼ (1) 프로젝트 폴더를 만들고,

```
mkdir webpack-n-babel
```

▼ (2) 프로젝트 경로로 이동하기

```
cd webpack-n-babel
```

▼ (3) 프로젝트 구조용 폴더도 만들어줍니다.

```
# src와 public 폴더를 만들어요.
# 더 많은 명령어가 궁금하다면, [쓰는 터미널 이름] commands로 검색해봅시다. :)
mkdir -p src public
```

▼ (4) 초기화를 해봅시다.

👉 프로젝트 초기화 후 vscode에서 폴더 구조를 보면 package.json 파일이 생긴 걸 확인할 수 있어요.
package.json은 프로젝트 정보를 정의하고 의존하는 패키지 버전 정보를 명시해두는 파일입니다. 우리가 프로젝트를 실행할 때 쓰는 yarn start의 start 같은 명령어도 이 파일 안에 다 명시해두고 사용하는거예요. 없으면 안되는 중요한 파일이란거, 감이 팍팍 오시나요? 😃

```
yarn init -y
```

▼ (5) webpack을 설치해요.

```
yarn add -D webpack webpack-cli
```

▼ (6) 명령어도 한 번 추가해볼까요!

▼ **[코드 스니펫] package.json**

```
"scripts": {
    "build" : "webpack --mode production",
}
```

▼ (7) webpack 설정을 해봅시다.

▼ **[코드 스니펫] webpack.config.js**

```
// webpack.config.js
const path = require("path");

module.exports = {
    mode: "development",
    entry: {
        main: './src/index.js'
    },
    output: {
```

```
        filename: '[name].bundle.js',
        path: path.resolve(__dirname, 'build')
    },
    resolve: { extensions: ["*", ".js", ".jsx"] },
    stats: { children: true }
};
```

▼ (8) 이제 html과 css를 위한 설정도 해볼까요!

▼ [코드 스니펫] 필요한 패키지 설치하기

```
# webpack으로 html 파일을 묶어주기 위해 필요한 패키지 설치
yarn add -D html-webpack-plugin clean-webpack-plugin

# webpack으로 css 파일을 묶어주기 위해 필요한 패키지 설치
yarn add -D mini-css-extract-plugin css-loader sass-loader file-loader
```

▼ [코드 스니펫] webpack.config.js에 설정 추가하기 - 1

```
const MiniCssExtractPlugin = require('mini-css-extract-plugin');
const HtmlWebpackPlugin = require('html-webpack-plugin');
const { CleanWebpackPlugin } = require('clean-webpack-plugin');
const webpack = require('webpack');
```

```
module: {
        rules: [
            {
                test: /\.(sa|sc|c)ss$/,
                use: [
                    {
                        loader: MiniCssExtractPlugin.loader,
                        options: {
                            hmr: true,
                            reloadAll: true
                        }
                    },
                    'css-loader',
                    'sass-loader'
                ]
            },
            {
                test: /\.(png|jpg|svg|gif)/,
                use: [
                    'file-loader'
                ]
            }
        ]
    },
plugins: [
        new CleanWebpackPlugin(),
        new HtmlWebpackPlugin({
            title: 'webpack-react-start-kit',
            template: './public/index.html'
        }),
        new MiniCssExtractPlugin({
            filename: '[name].css',
            chunkFilename: '[id].css'
          }),
    ]
```

▼ [코드 스니펫] webpack.config.js에 설정 추가하기 - 2

```
const MiniCssExtractPlugin = require('mini-css-extract-plugin');
const HtmlWebpackPlugin = require('html-webpack-plugin');
const { CleanWebpackPlugin } = require('clean-webpack-plugin');
const webpack = require('webpack');
```

```
module: {
        rules: [
            {
                test: /\.(sa|sc|c)ss$/,
                use: [
                    {
                        loader: MiniCssExtractPlugin.loader,
                        options: {
                            hmr: true,
                            reloadAll: true
                        }
                    },
                    'css-loader',
                    'sass-loader'
                ]
            },
            {
                test: /\.(png|jpg|svg|gif)/,
                use: [
                    'file-loader'
                ]
            }
        ]
    },
plugins: [
        new CleanWebpackPlugin(),
        new HtmlWebpackPlugin({
            title: 'webpack-react-start-kit',
            template: './public/index.html'
        }),
        new MiniCssExtractPlugin({
            filename: '[name].css',
            chunkFilename: '[id].css'
          }),
    ]
```

▼ html 파일 만들기

👉 public 폴더 아래에 index.html을 만들어봅시다.
CRA로 만들었던 프로젝트에서 일부만 복사해볼거예요. 😊

▼ [코드 스니펫] - public/index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />

    <title>React App</title>
```

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

▼ 17) babel

👉 babel은 컴파일러(Compiler)입니다. 단순히 말하자면 번역기예요.
우리가 작성할 ES6, 혹은 그 이상 버전의 자바스크립트와 JSX는 사실 브라우저가 지원해줘야만 실행할 수 있어요. 브라우저가 지원해주지 않는다면 당연히 코드도 동작하지 않아요. 😢
babel은 브라우저가 지원하지 않아도, 해당 브라우저가 알아 들을 수 있도록 번역해 준답니다.

▼ (1) 바벨 설치하기

```
# 바벨을 쓸 때 필요한 라이브러리는 내가 어떤 걸 해볼지에 따라 차이가 날 수 있어요.(사실 웹팩도, 다른 패키지들도 모두 그렇죠!)
# 이번에는 정말 꼭 필요한 것들만 설치해서 써봅니다. 저는 아래처럼 구성해볼거예요.
# @babel/core : 바벨 코어 라이브러리
# babel-loader : webpack에 babel 적용하기 위한 라이브러리
# @babel/preset-env : es6를 es5로 컴파일링 해주는 라이브러리
# @babel/preset-react : JSX를 자바스크립트 코드로 변환해주는 라이브러리
yarn add -D @babel/core babel-loader @babel/preset-env @babel/preset-react
```

▼ (2) babel 설정하기

```
// .babelrc
{
  "presets": ["@babel/preset-env", "@babel/preset-react"]
}
```

▼ 18) webpack과 babel 연동하기

👉 이제 webpack과 babel을 연동해주면 기본적인 설치와 설정이 모두 끝납니다!

▼ (1) webpack과 babel 연동하기

```
// webpack.config.js
...
module: {
    rules: [
      ...
      {
        test: /\.(js|jsx)$/,
        exclude: /node_modules/,
        use: {
          loader: "babel-loader",
        },
      },
    ],
  },
```

▼ 19) 리액트 프로젝트 띄우기

> 👉 기본적인 설치와 설정이 끝났습니다. 이제 프로젝트 띄우냐고요?
> 아니요! 이제 React를 설치해야죠! 리액트를 설치해야 리액트를 쓰니까요. 😉
> 네? 너무 번거롭다고요? ㅎㅎ 곧 끝납니다! 화이팅! ~~여러분은... CRA를 소중히 하지 않았지요...~~

▼ (1) 리액트 설치하기

```
yarn add react react-dom
```

▼ (2) index.js 만들기

```
import React from "react";
import ReactDOM from "react-dom";

ReactDOM.render(<div>안녕 여러분! :)</div>, document.getElementById("root"));
```

▼ (3) webpack-dev-server 설치하기

```
yarn add webpack-dev-server
```

▼ (4) webpack 설정 추가하기

> 👉 설정은 웹팩 공식 문서를 참고해서 해주면 됩니다. 😊
> 버전에 따른 변화가 잦기 때문에, 혹시 오류가 나온다면 꼭 공식 문서의 설정과 내 설정을 비교해보세요. (공식 문서 링크 →)

```
// webpack.config.js
...
module.exports = {
  ...
  devtool: "inline-source-map",
  devServer: {
    static: {
      directory: path.join(__dirname, 'public'),
    },
    compress: true,
    port: 9000,
  },
  plugins: [
    ...
    new webpack.HotModuleReplacementPlugin(),
  ],
};
```

▼ (5) 시작 커멘드 추가하기

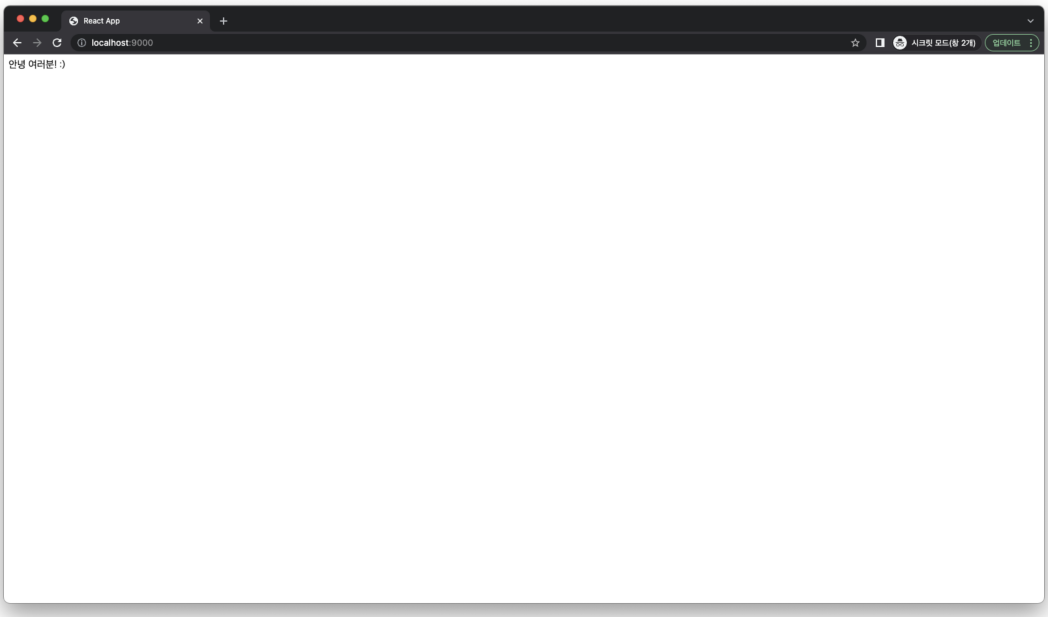
```
// package.json
"scripts": {
```

```
    ...
    "start": "webpack-dev-server"
},
```

▼ (6) 브라우저에서 확인해보자!

👉 프로젝트 시작 후 localhost:9000으로 들어가봅시다! 아래 같은 이미지가 뜬다면 프로젝트 설정 완료!

```
yarn start
```



08. Virtural DOM

▼ 20) 가상 돔이란?

👉 렌더링 엔진 동작과정 :
- 파싱 단계
- 렌더 트리 단계
- 레이아웃 단계
- 페인트 단계

- DOM 트리 중 하나가 수정될 때마다 모든 DOM을 뒤지고, 수정할 걸 찾고, 싹 수정을 한다면?
  → 필요없는 연산이 너무 많이 일어난다!
  → 그래서 등장한 게 가상돔!

- 가상돔은 메모리 상에서 돌아가는 가짜 DOM입니다.
- 가상돔의 동작 방식: **기존 DOM과 어떤 행동 후 새로 그린 DOM(가상 돔에 올라갔다고 표현합니다)을 비교해서 정말 바뀐 부분만 갈아끼워줍니다! → 돔 업데이트 처리가 정말 간결하죠!**

> 💡 알면 덜 찜찜한 이야기: **DOM이 정말 그렇게 느려?**
> 반은 맞고 반은 틀려요. DOM은 사이트 구조에 따라 가상돔을 쓰는 것보다 훨씬 성능이 좋을 수 있고(=빠를 수 있고), 속이 터지게 느릴 수 있습니다.

09. 함수형 컴포넌트 - (1) 함수형 컴포넌트 살펴보기

▼ 21) 함수형 컴포넌트 살펴보기

> 👉 Component란?
> 감잡을 때 이렇게 배웠어요 : **React가 레고**라면 **Component는 블록**입니다!
> 여러분은 이제 이렇게 기억하세요 : **독립적인 기능을 수행**할 수 있는, **재사용이 가능한 최소 단위**

▼ 함수형 컴포넌트 생김새

```
function IAmComponent(props) {
  return (<p>I am component! :) Functional component!</p>);
}
```

▼ 22) 첫 컴포넌트 만들기

> 👉 함수형 컴포넌트를 만들고 App.js에 가져와서 써봅시다.

▼ 함수형 컴포넌트 만들기

```
//MyComponents.js
const One = () => {
  return (
    <div>
      <h1>One</h1>
    </div>
  )
}

const Two = () => {
  return (
    <div>
      <h1>Two</h1>
    </div>
  )
}

export default One;
export {Two};
```

▼ App.js에서 불러오기

```
//App.js
import One, { Two } from './MyComponents';

function App() {
  return (
    <div className="App">
      <One/>
      <Two/>
    </div>
  );
}

export default App;
```

10. 함수형 컴포넌트 - (2) 함수형 컴포넌트 상태관리

▼ 23) state와 props

👉 컴포넌트가 어떻게 데이터를 관리할까요?
컴포넌트의 데이터 관리를 상태 관리라고 해요.

- props

👉 **props**는,
1. Component가 부모 Component로부터 받아온 데이터입니다.
2. 읽기 전용이라 props는 절대절대 수정하면 안됩니다!
3. props는 순수 함수처럼 동작해야 합니다.

  - 순수 함수
    - 받아온 값(인자)를 수정하지 않고,
    - 항상 같은 값을 넣으면 동일한 결과를 내주는 함수

```
const IamPureFunction = (a, b) => {
  return a+b;
}

// a, b가 변하지 않았고, 1과 2를 넣으면 언제나 3이 나온다!
IamPureFunction(1, 2);

// a가 바뀌었으니 순수하지 않다!
// 이외에도 랜덤 함수를 사용해 랜덤한 값을 리턴해도 순수하지 않다!
const IamNotPureFunction = (a, b) => {
  a = b-a;
}

IamNotPureFunction(5, 9);
```

- state

👉 **state**는,
1. Component가 가지고 있는 데이터입니다.
2. 함수형 컴포넌트는 useState() 훅을 사용해서 상태값을 가질 수 있습니다.
3. state는 직접 수정해선 안됩니다!
4. state는 비동기적으로 업데이트 됩니다.

- useState()

▼ 미리 써보는 리액트 훅

```
// MyComponent.js
import React from "react";

const One = () => {

  const [name, setName] = React.useState("mean0");

  return (
    <div>
      <h1>One</h1>
      <p>{name}</p>
      <button onClick ={() => {
        setName("perl");
      }}>change!</button>
    </div>
  )
}

const Two = () => {
  return (
    <div>
      <h1>Two</h1>
    </div>
  )
}

export default One;
export {Two};
```

11. 컴포넌트가 돔에 그려지려면?

👉 앞선 강의 함수형 컴포넌트를 만들어봤습니다. 😊
함수형 컴포넌트가 return 해주는 엘리먼트가 화면에 표시되는 것도 확인했지요. 이 엘리먼트들을 React 엘리먼트라고 해요. 그럼, React 엘리먼트가 어떻게 화면에 표시되는 건지도 한 번 살펴봅시다.

[참고]
17.0.2버전에서는 `ReactDOM.render(element, container[, callback])`를 사용했어요.
(링크→)

▼ 24) ReactDOM.createRoot()

- pubilc/index.html을 열고 `<div id="root"></div>`를 찾아봅시다.
  - 이걸 루트 DOM 노드라고 불러요.
  - 우리가 만든 모든 컴포넌트는 이곳에 들어가게 될 겁니다.
    !주의! 물론 예외가 있습니다. 나중에 다뤄볼 Portal인데요, Portal을 사용할 부분을 제외한 모든 컴포넌트는 루트 돔 노드에 들어갈거예요.
- src/index.js를 열고 루트를 어떻게 만드는 지 확인해봅시다.
  - `const root = ReactDOM.createRoot(document.getElementById('root'));`

▼ 25) ReactDOM.createRoot(someElement).render()

- render()는 루트 돔 노드에 리액트 엘리먼트를 전달하는 역할을 합니다. (화면에 띄워주죠!)
- 리액트 엘리먼트는 기본적으로는 **불변 객체**입니다.
  - 속성, 자식 등 엘리먼트의 내용을 변경할 수 없습니다.
- render()는 엘리먼트에 변화가 필요할 경우 필요한 부분만 업데이트 해주는 역할을 합니다.

12. 끝 & 숙제 설명

우리 이번 주차에 굉장히 많은 내용을 배웠어요! 강의에서 배운 내용을 잘 정리해서 숙제를 해봅시다!
1. 새로운 프로젝트를 만들고,
2. TIL 목록이 나오는 부분과,
3. 과목, 내용, 공부시간을 입력하는 부분을 만들어주세요.
4. [추가하기]를 누르면 목록에 추가되도록 해주세요.

▼ 예시 화면

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/95eef48d-d6b6-4325-a07e-fa306366c2c6/homework_w1.zip