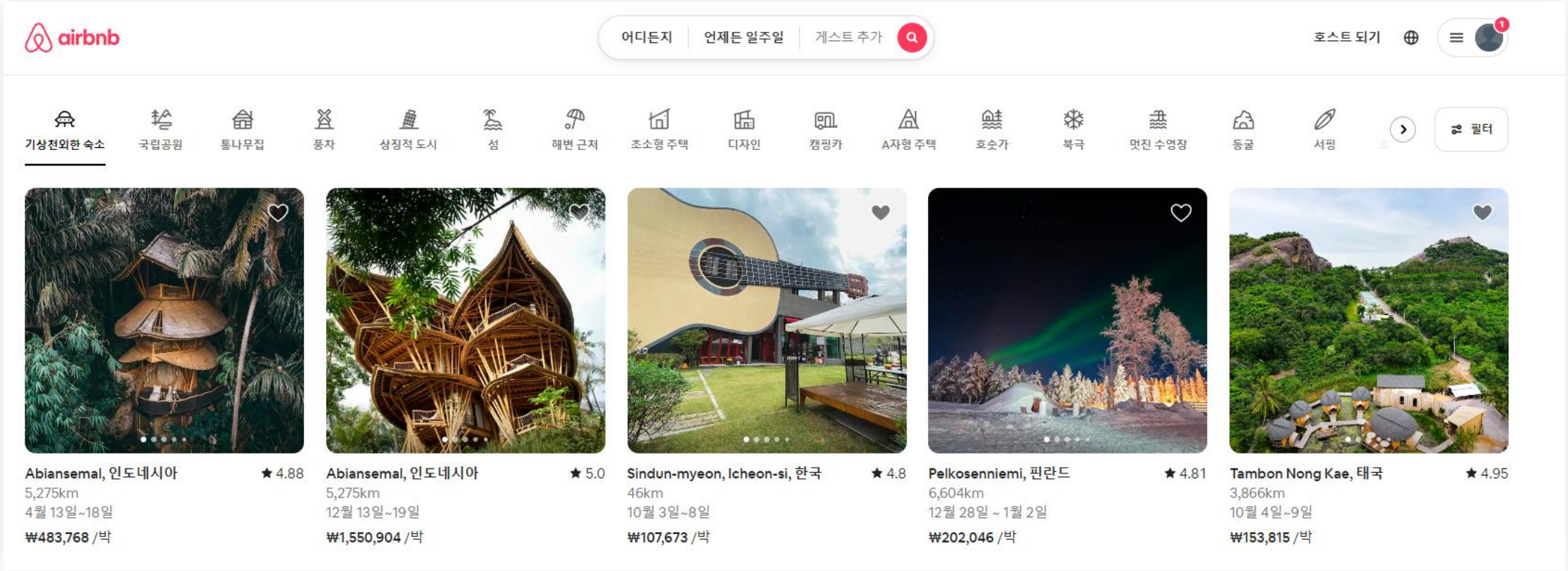
# React

## Component와 JSX

# Component

# Component

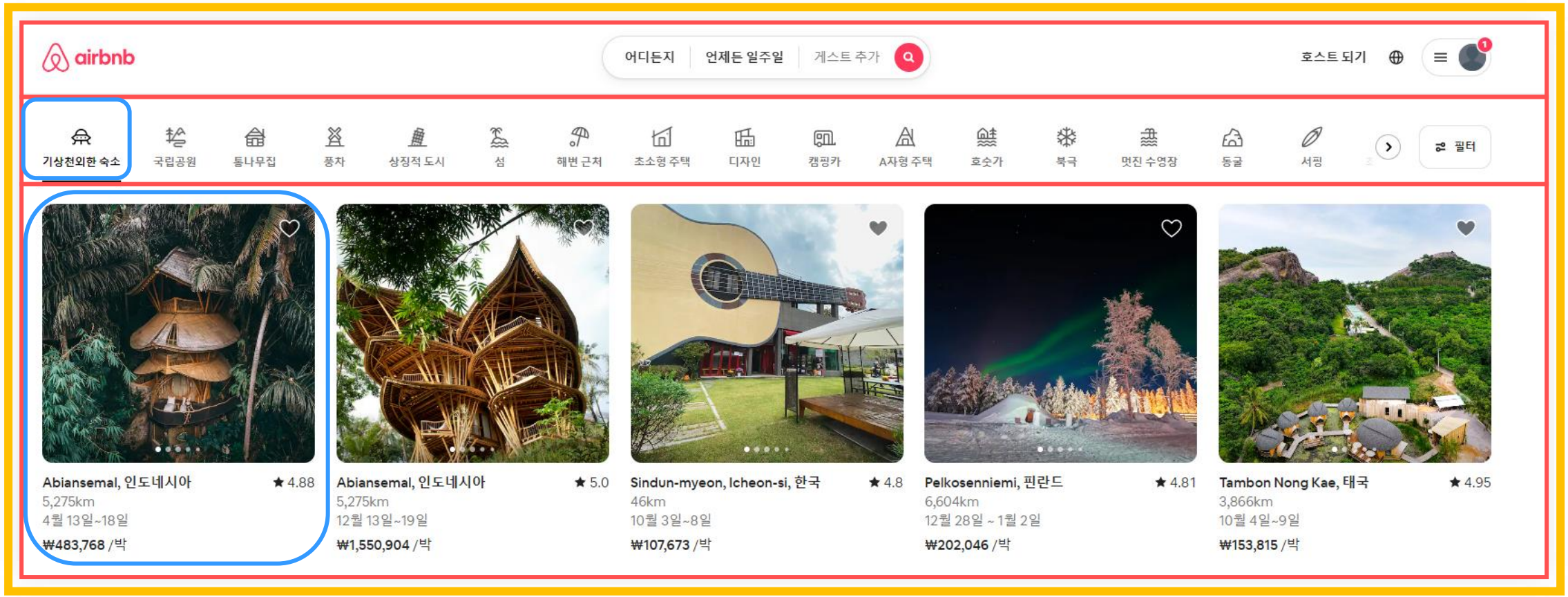

airbnb
어디든지
언제든 일주일
게스트 추가
호스트 되기
기상천외한 숙소
국립공원
통나무집
풍차
상징적 도시
섬
해변 근처
초소형 주택
디자인
캠핑카
A자형 주택
호숫가
북극
멋진 수영장
동굴
서핑
필터
Abiansemal, 인도네시아 ★ 4.88
5,275km
4월 13일~18일
₩483,768 /박
Abiansemal, 인도네시아 ★ 5.0
5,275km
12월 13일~19일
₩1,550,904 /박
Sindun-myeon, Icheon-si, 한국 ★ 4.8
46km
10월 3일~8일
₩107,673 /박
Pelkosenniemi, 핀란드 ★ 4.81
6,604km
12월 28일 ~ 1월 2일
₩202,046 /박
Tambon Nong Kae, 태국 ★ 4.95
3,866km
10월 4일~9일
₩153,815 /박

# Component란?

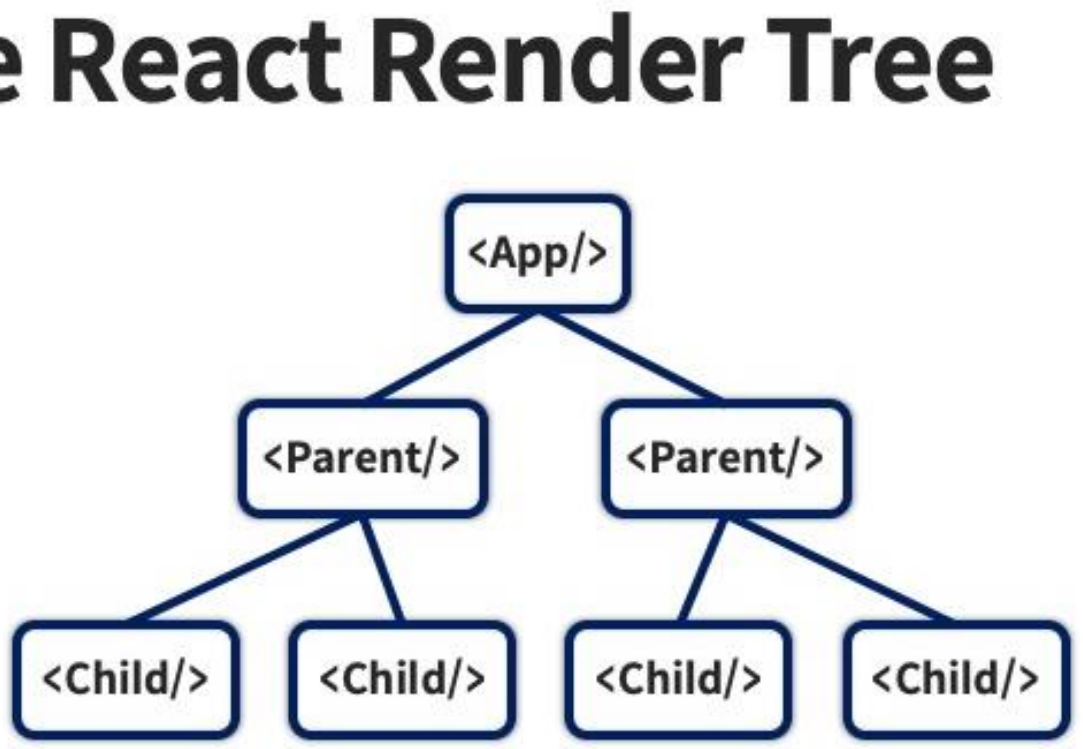
- React의 꽃이라 불리는 React의 핵심
- 화면을 구성하는 하나의 부분
- 내부의 데이터만 변경해서 전체적인 틀(UI)을 재사용 가능
- 데이터(props)를 입력 받아 View 상태(state)에 따라 DOM Node를 호출한다.
- UI를 **재사용 가능한 개별적인 여러 조각**으로 나누고, 각 조각을 개별적으로 나누어 분리 가능하다.

# Component

1. 재사용 되고 있는 부분을 담당하고 있는 작은 단위의 컴포넌트
2. 페이지의 부분(section, header 등)을 담당하고 있는 컴포넌트
3. 페이지 전체를 담당하고 있는 컴포넌트



# Component 트리 구조



# Component 종류

- **함수형 Component**
  - 짧고 직관적
  - Vanilla JS와 같은 기본적인 function 구조를 이용해 더 직관적이며 추상적
  - 메모리 자원을 덜 사용한다.

- **클래스형 Component**
  - State와 라이프 사이클 기능 이용 가능
  - Render 함수 필수
  - 컴포넌트를 만들 때 기존 Components 클래스에서 상속받아서 사용

# Component 를 만드는 방법

- 클래스형 컴포넌트 vs. 함수형 컴포넌트
  - React 초창기에는 함수형 컴포넌트에는 현재 리액트의 핵심기능(state, lifecycle..)등 기능을 사용할 수 없었어요.
  - React 16.8 부터 hooks 의 등장으로 함수형 컴포넌트에도 핵심기능을 사용할 수 있게 되었습니다.
  - 과거에는 다양한 기능을 사용할 수 있는 클래스형 컴포넌트를 사용했지만, 현재는 더 쉬운 문법을 가지고 있는 함수형 컴포넌트의 사용 비율이 압도적으로 높아요!
  - 공통적으로 컴포넌트의 이름은 PascalCase 사용!

클래스형 컴포넌트를 만드는 방법도 배울거지만, 저렇게 쓰는구나 ~ 하고 알아만 주시면 됩니다

# 클래스형 컴포넌트

```jsx
import { Component } from "react";

class ClassComponent extends Component {
    render() {
        const classss = 'kdt';
        return (
          <>
            <div>{classss == "kdt" ? "kdt 반가워요" : "누구..."}</div>
            <div>반가워!</div>
          </>
        )
    }
}

export default ClassComponent;
```

* export: 내보내기

# 함수형 컴포넌트

```jsx
function FuncComponent() {
    const classss = "kdt";
    return (
        <>
        <div>{classss == "kdt" ? "kdt 반가워요" : "누구..."}</div>
        <div>반가워!</div>
        </>
    );
}
export default FuncComponent;
```

* export: 내보내기

```jsx
const FuncComponent = () => {
    const classss = "kdt";
    return (…
    );
}
export default FuncComponent;
```

# 컴포넌트 import

* import: 불러오기

```
import ClassComponent from './ClassComponent';
import FuncComponent from './FuncComponent';

function App(){
    return (
      <>
        <ClassComponent></ClassComponent>
        <FuncComponent />
      </>
    )
}

export default App;
```

# 복습

| Syntax | Export statement | Import statement |
| --- | --- | --- |
| Default | `export default function Button() {}` | `import Button from './Button.js';` |
| Named | `export function Button() {}` | `import { Button } from './Button.js';` |

# JSX

# JSX

JavaScript + XML

- 자바스크립트 확장 문법으로 XML 과 유사
- 바벨을 이용해 일반 자바스크립트 코드로 변환

# JSX 1. 전체는 하나의 태그로 감싸야

html 태그는 항상 return 이후에, 반드시 하나의 부모 요소가 전체 요소를 감싸는 형태로!

+ 딱 하나의 태그만 return할 때는 ()가 필요 없지만 여러 개의 태그가 있다면 return (~) 괄호 내부에 작성해야 해요!

```
function App() {
  const str = 'hello';

  return (
    <div className='App'>
      <span>{str}</span>
      <span>world</span>
    </div>
  );
}
export default App;
```

```
function App() {
  const str = 'hello,';

  return (
      <span>{str}</span>
      <span>world!!</span>
  );
}
export default App;
```

JSX expressions must have one parent element.

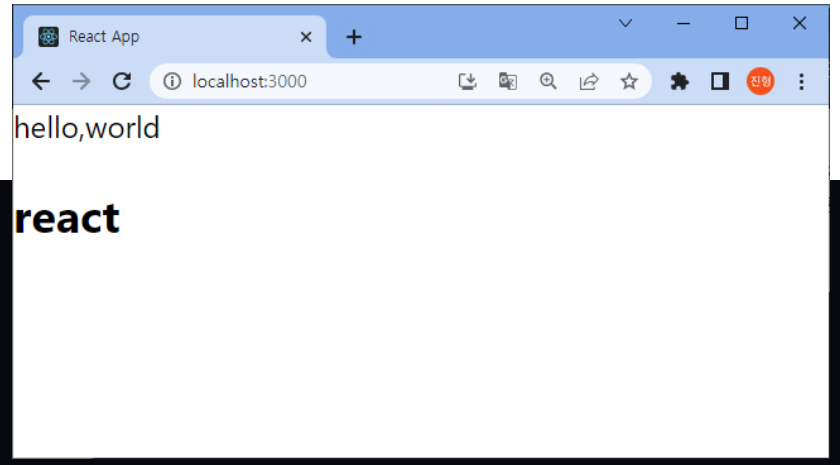
# JSX 2. html with JS

html를 작성하다가 중간에 js 문법을 사용하고 싶을 때는 {중괄호}로 감싸야 합니다.


```
const str = 'hello,';
return (
    <div className='App'>
      <span>{str}</span>
      <span>{str === 'hello,' ? 'world' : '세상'}</span>
      <div>
        {str === 'hello,' ? (<div><h2>react</h2></div>) : (<div><h2>리액트</h2></div>)}
      </div>
    </div>
  );
```

# JSX 2. html with JS

html를 작성하다가 중간에 js 문법을 사용하고 싶을 때는 {중괄호}로 감싸야 합니다.




```
const str = 'hello,';
return (
    <div className='App'>
      <span>{str}</span>
      <span>{str === 'hello,' ? 'world' : '세상'}</span>
      <div>
        {str === 'hello,' ? (<div><h2>react</h2></div>) : (<div><h2>리액트</h2></div>)}
      </div>
    </div>
  );
```

# JSX 2. html with JS

JSX 삼항연산자 대신 if 문이 올 수 없어요,

+ for문또한 JSX 내부에 올 수 없어요!

(for문과 if 문을 사용하고 싶다면 return 이 오기 전에 결과값을 저장하고 사용)

for, if 등이 오려면
특정 변수에 저장하고 사용해야 해요

```
const numbers = [1, 2, 3, 4];
let sum = 0;
for (let num of numbers) {
  sum += num;
}

export default function App() {
  return <span>{sum}</span>;
}
```

삼항연산자 대신 if 문이 나올 수 없어요!

```
const str = 'hello,';
return (
    <div className='App'>
      <span>{str}</span>
      <span>{str === 'hello,' ? 'world' : '세상'}</span>
      <div>
        {str === 'hello,' ? (<div><h2>react</h2></div>) : (<div><h2>리액트</h2></div>)}
      </div>
    </div>
  );
```

# JSX 3. 인라인 style 적용

- CSS 를 인라인 형태로 적용할 때는 {object}형태로 저장해야 해요!
- CSS 속성은 dash-case → camelCase

```
<div style="font-size: 32px; background-color: crimson;">인라인 스타일</div>
```

```
export default function App() {
  return <div style={{ fontSize: '32px', backgroundColor: 'crimson' }}>인라인 스타일</div>;
}
```

- return 위에 object를 변수에 담아놓고 사용할 수도 있어요.

```
export default function App() {
  const divStyle = { fontSize: '32px', backgroundColor: 'crimson' };
  return <div style={divStyle}>인라인 스타일</div>;
}
```

# JSX 4. class와 onclick

class 대신 className, onclick대신 onClick 을 사용합니다.

```
return (
    <div className='App' onClick={()=>{alert('hi')}}>
      <span>{str}</span>
      <span>world!!</span>
    </div>
  );
```

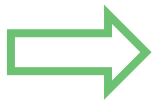
- onClick={클릭되었을 때 실행할 JS 코드}
- className='클래스 이름'
- 클래스 이름도 따로 선언해서 className={class-name}처럼 사용할 수 있어요

# JSX 5. closing tag

opening tag(빈태그)도 closing tag가 필요해요!

```
export default function App() {
  return (
    <div>
      <input type="text">
    </div>
  );
}
```

```
JSX element 'input' has no corresponding closing tag.
```



```
export default function App() {
  return (
    <div>
      <input type='text' />
      <br />
      <img src={~~} alt='alt' />
    </div>
  );
}
```

```
export default function App() {
  return (
    <div>
      <input type='text'></input>
      <br></br>
      <img src={~~} alt='alt'></img>
    </div>
  );
}
```

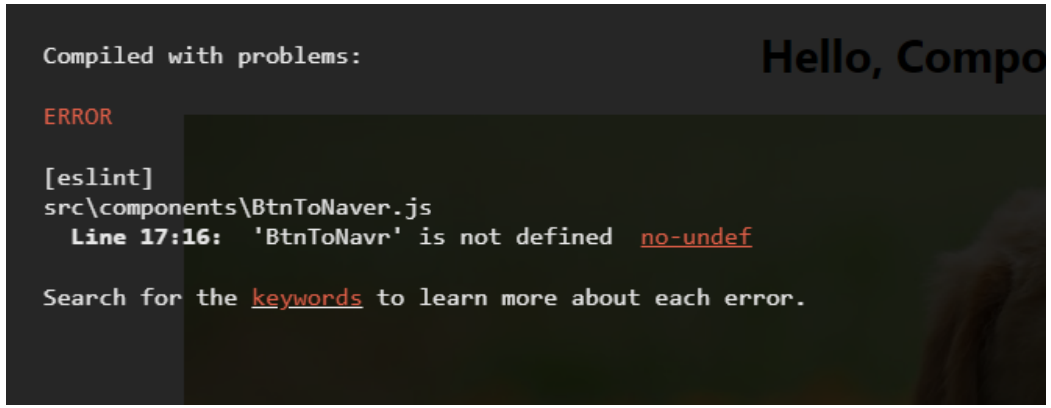
# JSX 6. 주석 사용

주석은 {/* 이렇게 씁니다!*/}

```
export default function App() {
  // return 밖에서는 기존의 Javascript 처럼
  /* 주석을 사용하면 돼요! */
  return (
    <div>
      주석 test!
      {/* return 내부에서주석사용은 이렇게 합니다 */}
      {/* 물론 기존의 ctrl + '/' 단축키 이용하면 됩니다. */}
    </div>
  );
}
```

# 리액트의 디버깅

- 기존 JavaScript 에서는 에러를 확인하기 위해서 console창으로 봐야하지만,
  React는 치명적인 버그일 경우에 화면에 바로 띄워줍니다.




- JS의 문제점을 보완하고자 strict mode 강제!
  - 어디서 실수 했는지까지 자세히 제공하니 에러 창을 두려워하지 말고 에러메세지를 잘 읽어주세요!