



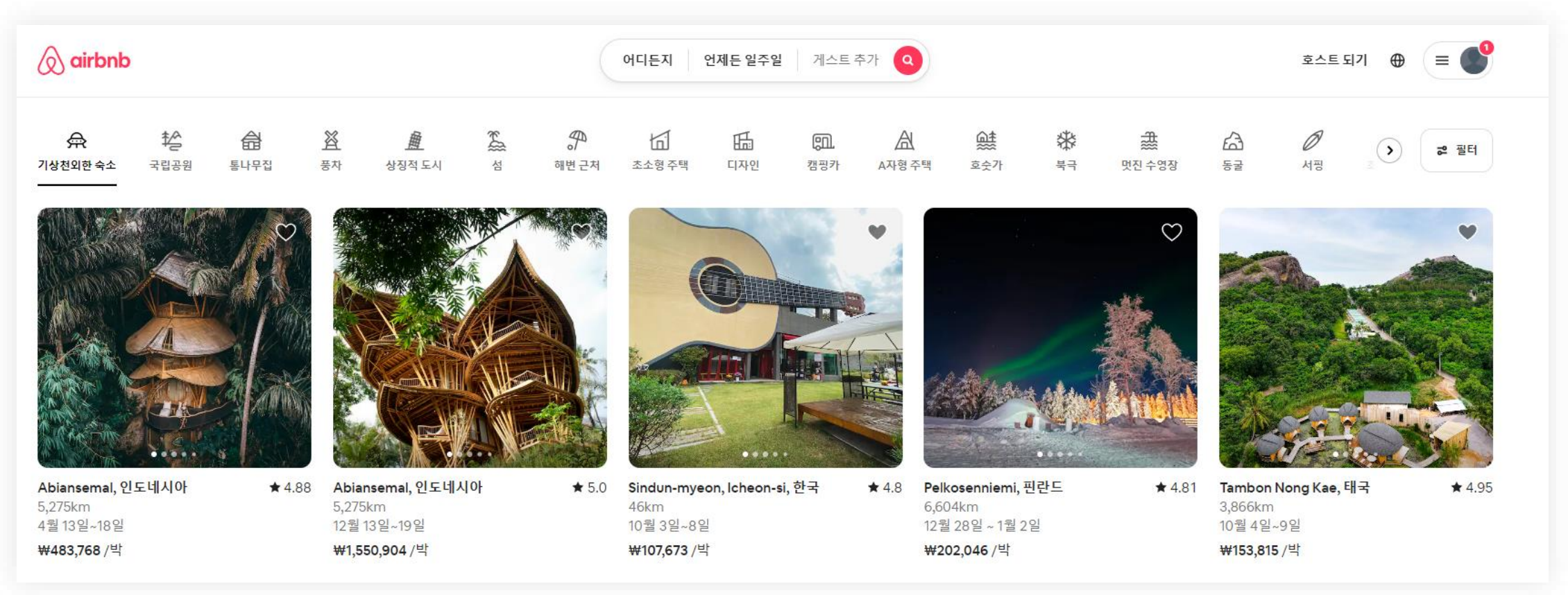
React

01. Component & JSX

- 클래스 컴포넌트와 함수형 컴포넌트 & JSX 문법 규칙 -

Compoment

Component

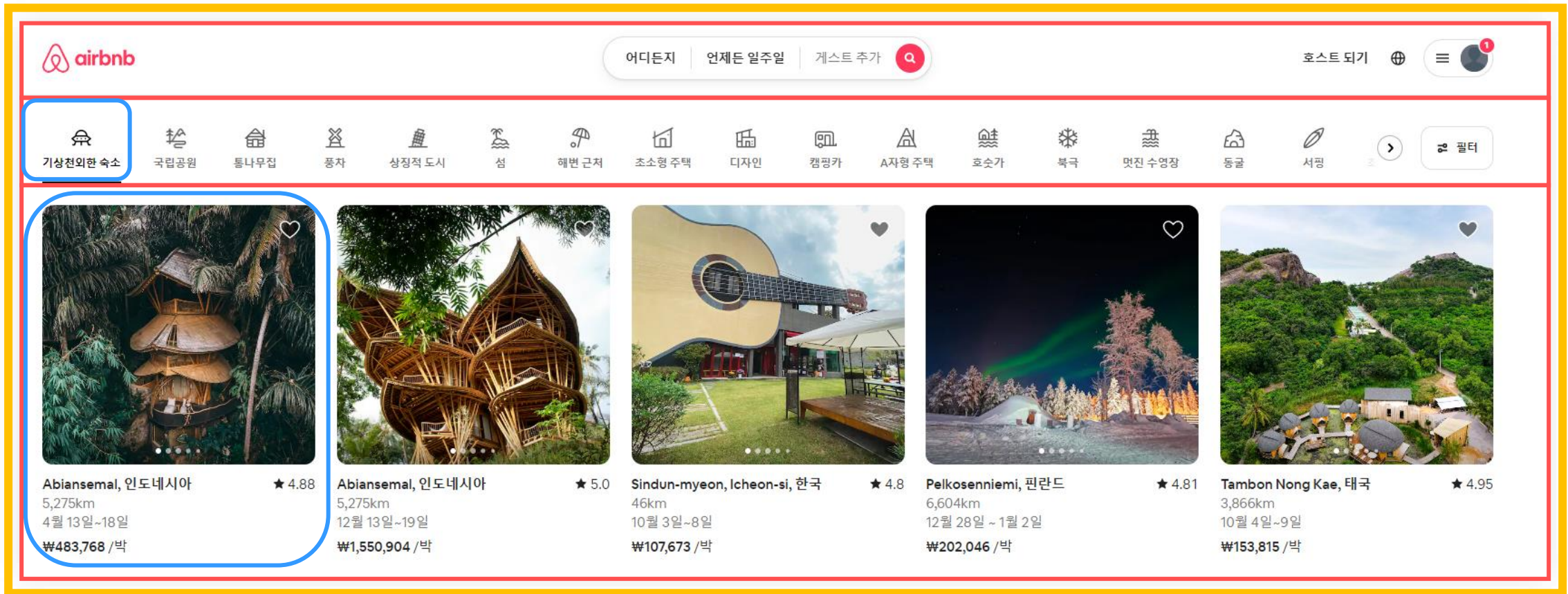


Component

- React의 꽃이라 불리는 **React의 핵심**
- React 에서 화면을 구성하는 하나의 부분
- 내부의 데이터만 변경해서 전체적인 틀(UI)을 재사용 가능
- UI를 재사용 가능한 개별적인 여러 조각으로 나누고, 각 조각을 개별적으로 나누어 분리 가능

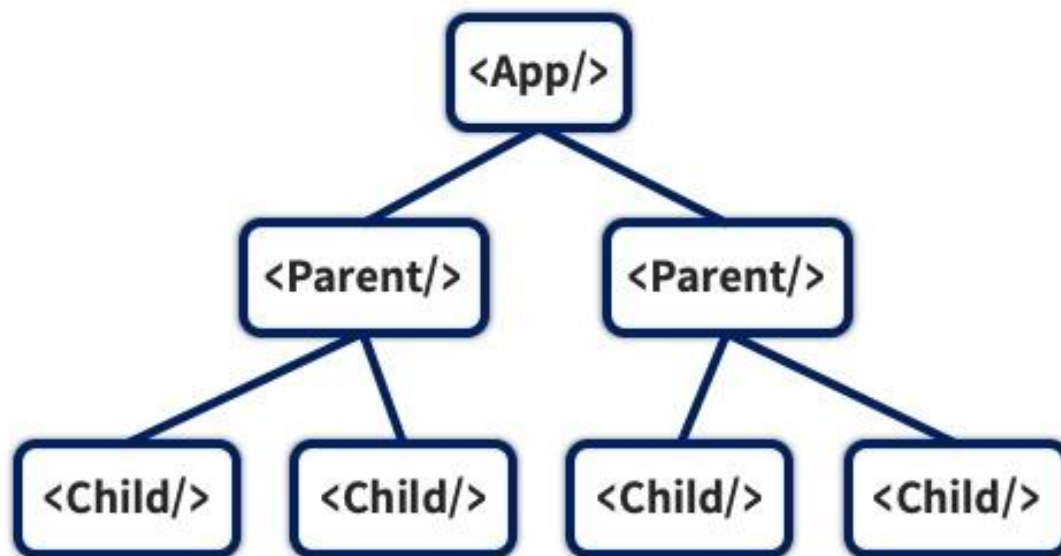
Component

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

The React Render Tree



React 컴포넌트 종류

- 클래스형 컴포넌트 (Class Component)
 - render 함수를 반드시 사용해야 함
 - lifecycle 기능
 - 컴포넌트를 만들 때 기존 Components 클래스에서 상속받아서 사용
- 함수형 컴포넌트 (Functional Component)
 - 짧고 직관적이며, 사용방법이 더 간단함
 - 메모리 사용에 이점이 있음

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

이렇게 생겼어요!

```
import React, {Component} from 'react';

class ClassComponent extends Component {
  render() {
    return(
      <h1>Class Component 입니다.</h1>
    );
  }
}
```

```
export default ClassComponent;
```

클래스형 컴포넌트

함수형 컴포넌트

```
export default const FunctionComponent = () =>
{
  return
    <div>
      Funtional Component 입니다
    </div>
};
```

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>
  );
}
```

JSX expressions must have one parent element.

```
export default App;
```

JSX 2. html with JS

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



return ();



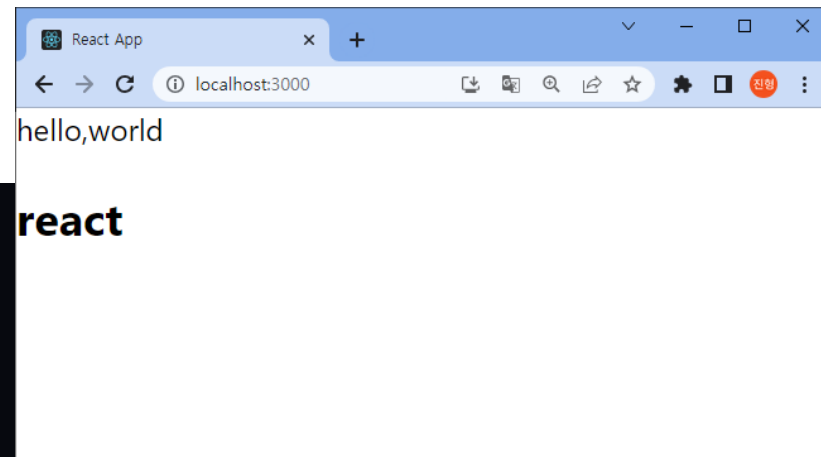
{ }

```
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,

+ for문 또한 JSX 내부에 올 수 없어요!

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

삼항연산자 대신 if 문이 나올 수 없어요!

for, if 등이 오려면
특정 변수에 저장하고 사용해야 해요

```
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>
);
```

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

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

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>
  );
}
```

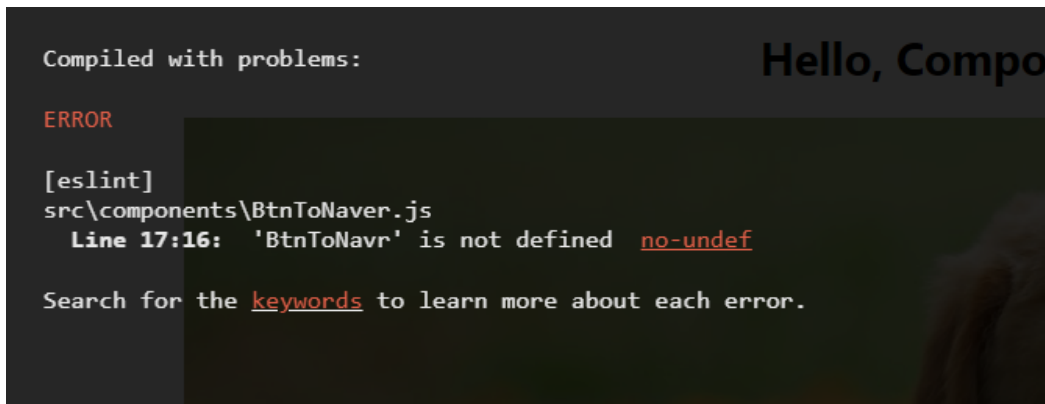
JSX 6. 주석 사용

주석은 `{/* 이렇게 씁니다!*/}`

```
export default function App() {  
  // return 밖에서는 기존의 Javascript 처럼  
  /* 주석을 사용하면 돼요! */  
  return (  
    <div>  
      주석 test!  
      { /* return 내부에서 주석 사용은 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!**