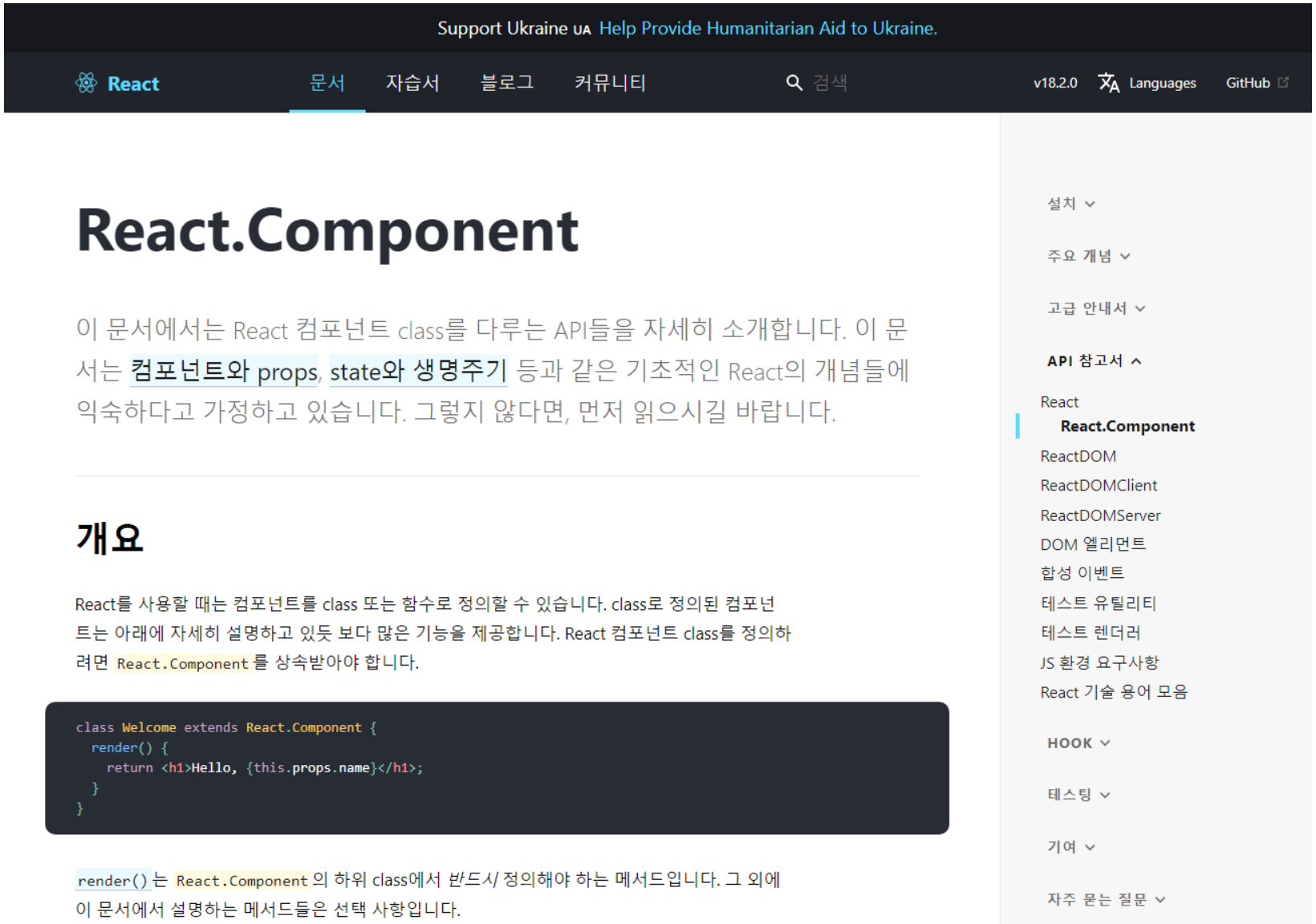
K-Digital Training KDT 풀스택 웹 개발자 양성 부트캠프 3기

React

WITH 팀 뤼쳐드

Component



React.Component

이 문서에서는 React 컴포넌트 class를 다루는 API들을 자세히 소개합니다. 이 문서는 컴포넌트와 props, state와 생명주기 등과 같은 기초적인 React의 개념들에 익숙하다고 가정하고 있습니다. 그렇지 않다면, 먼저 읽으시길 바랍니다.

개요

React를 사용할 때는 컴포넌트를 class 또는 함수로 정의할 수 있습니다. class로 정의된 컴포넌트는 아래에 자세히 설명하고 있듯 보다 많은 기능을 제공합니다. React 컴포넌트 class를 정의하려면 `React.Component` 를 상속받아야 합니다.

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

`render()` 는 `React.Component` 의 하위 class에서 *반드시* 정의해야 하는 메서드입니다. 그 외에 이 문서에서 설명하는 메서드들은 선택 사항입니다.

https://ko.reactjs.org/docs/react-component.html

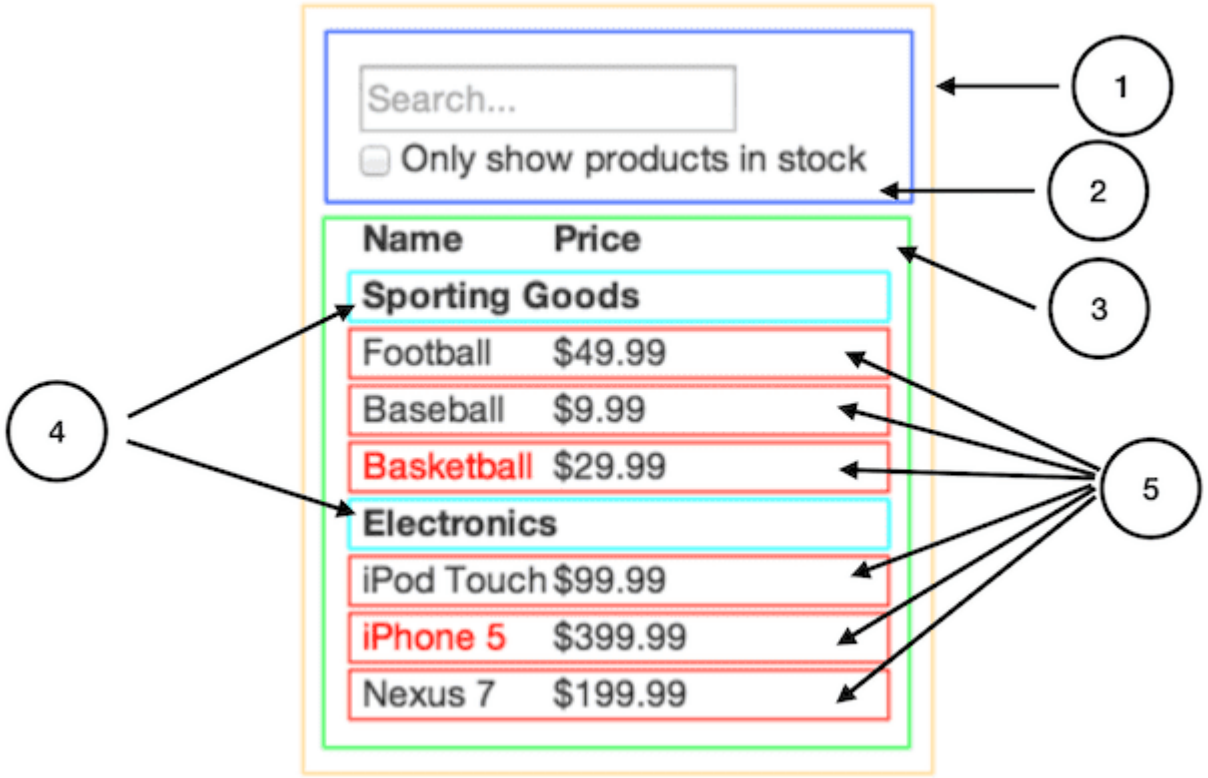
Component란?

- React의 꽃이라 불리는 React의 핵심
- MVC **View를 독립적으로 구성**해 재사용할 수 있고, 새로운 컴포넌트도 만들 수 있다.
- 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 코딩이 가능하다.

Component란?


SpringBoot
[스프링부트 (1)] 스프링부트 시작하기 (SpringBoo..
[스프링부트 (1)] 스프링부트 시작하기 (SpringBoot 프로젝트 설정 방법) 안녕하세요. 갓대희 입니다. 이번 포스팅은 [스프링 부트 기초, 스..
2020.04.09
[스프링부트 (12)] SpringBoot 에러 페이지 설정(C..
[스프링부트 (12)] SpringBoot 에러 페이지 설정 (Custom Error Page) 안녕하세요. 갓대희 입니다. 이번 포스팅은 [Spring Boot Custom Er..
2020.03.04
[스프링부트 (11)] SpringBoot YAML 적용하기(pr..
[스프링부트 (11)] SpringBoot YAML 적용하기 (properties vs yaml) 안녕하세요. 갓대희 입니다. 이번 포스팅은 [Spring Boot Properties..
2020.03.02
[스프링부트 (10)] SpringBoot Test(3) - 단위 테스..
[스프링부트 (10)] SpringBoot Test(3) - 단위 테스트(@WebMvcTest, @DataJpaTest, @RestClientTest 등) 안녕하세요. 갓대희 입니다...
2020.02.29
[스프링부트 (9)] SpringBoot Test(2) - @SpringB..
[스프링부트 (9)] SpringBoot Test(2) - @SpringBootTest로 통합테스트 하기 안녕하세요. 갓대희 입니다. 이번 포스팅은 [스프링 부트 통..
2020.02.28
[스프링부트 (8)] SpringBoot Test(1) - Junit 설정..
[스프링부트 (8)] SpringBoot Test(1) - Junit 설정 및 실행 안녕하세요. 갓대희 입니다. 이번 포스팅은 [스프링 부트 테스트 시작하기 (Junit..
2020.02.25
글 리스트 컴포넌트
글 결과 리스트 컴포넌트
글 목록 컴포넌트
결과 컴포넌트

Component란?



1 – 전체

2 – 입력란

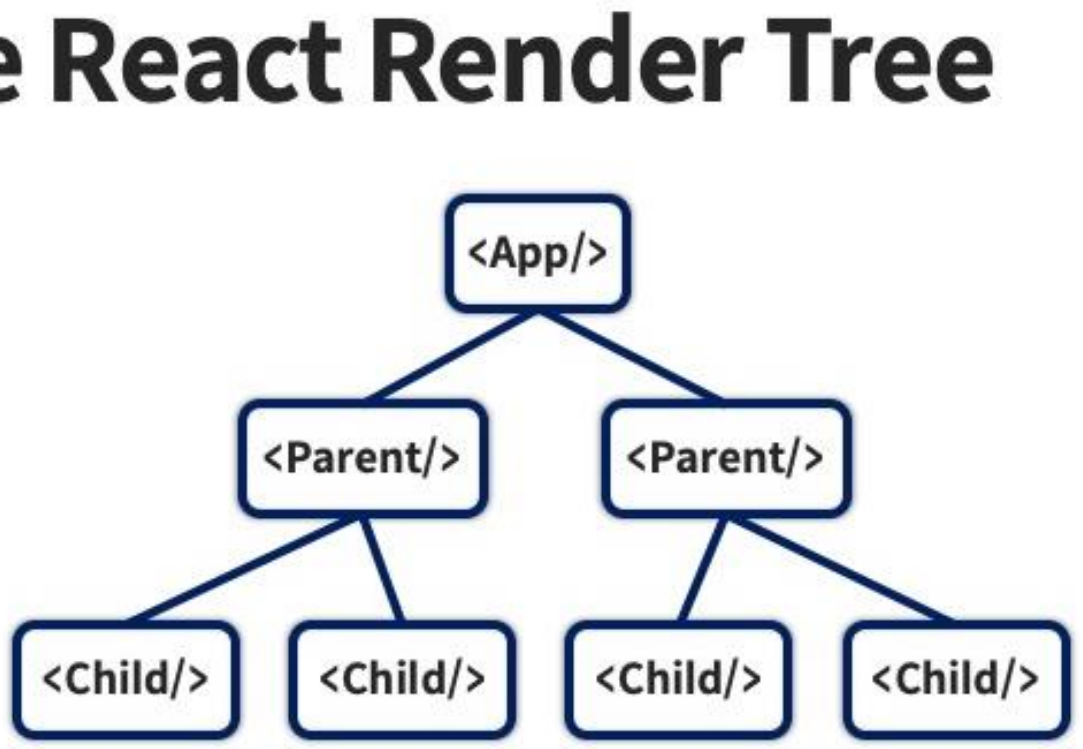
3 – 입력 결과표

4 – 카테고리 헤더

5 – 각 제품에 해당하는 행

https://ko.reactjs.org/docs/thinking-in-react.html#step-1-break-the-ui-into-a-component-hierarchy

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

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;
```

props

props란?

- properties 를 줄인 표현으로 컴포넌트 **속성**을 설정할 때 사용하는 요소
- props는 컴포넌트끼리 값을 전달하는 수단
- **상위 컴포넌트에서 하위 컴포넌트로 전달 (단방향 데이터 흐름)**

```
{/* 일반 사용법 */}
<ClassComponent></ClassComponent>

{/* props 사용법 */}
<ClassComponent title="제목" content="내용"></ClassComponent>
```

함수형 컴포넌트 props

- 부모 컴포넌트에서 전달한 props는 함수형 컴포넌트에서 **함수의 파라미터로 전달**받으며, **JSX 내부에서 { } 기호로 감싸서** 사용한다.

```
<FuncComponent name="코딩온"></FuncComponent>
```

* 부모 컴포넌트에서 name props 전달

```
const FuncComponent = (props) => {
    return (
        <>
        <div>안녕? {props.name}</div>
        <div>반가워!</div>
        </>
    );
}
```

* 자식 컴포넌트에서 name props 받음

defaultProps

- 부모 컴포넌트에서 props가 전달되지 않았을 때 **기본값으로 보여줄 props를 설정**하는 것

```
<FuncComponent ></FuncComponent>
```

* 부모 컴포넌트

```
const FuncComponent = (props) => {
    return (
        <>
        <div>안녕? {props.name}</div>
        <div>반가워!</div>
        </>
    );
}

FuncComponent.defaultProps = {
    name: '홍길동'
}
```

* 자식 컴포넌트

props.children

- 부모 컴포넌트에서 자식 컴포넌트를 호출할 때 태그 사이에 작성한 문자열

```
<FuncComponent name="코딩온">자식 내용</FuncComponent>
```

* 부모 컴포넌트

```
const FuncComponent = (props) => {
    return (
        <>
        <div>안녕? {props.name}</div>
        <div>반가워!</div>
        <h4>{props.children}</h4>
        </>
    );
}
```

* 자식 컴포넌트

propTypes

- 컴포넌트의 **필수 props를 지정**하거나 **props의 타입을 지정**할 때 사용
- JavaScript의 **"유연한 특성" 을 해결**하기 위해 권장되는 기능
- 정해진 타입이 아닌 다른 타입으로 정보가 전달될 시, 제대로 동작은 하지만 console에 오류가 나온다.

```
import PropTypes from "prop-types";
```

```
FuncComponent.propTypes = {
    name: PropTypes.string
}
```

클래스형 컴포넌트 props

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

클래스형 컴포넌트 props

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

    static defaultProps = {
        name: "기본 이름"
    };

    static propTypes = {
        name: PropTypes.string
    };
}
```

or

```
ClassComponent.defaultProps = {
    name: '홍길동'
}

ClassComponent.propTypes = {
    name: PropTypes.string
}
```