섹션3 컴포넌트, JSX, 속성, 상태

컴포넌트

리액트 어플리케이션은 결국 컴포넌트를 결합하여 만들어지는 것

컴포넌트 == html + css +js

컴포넌트 개념이 없다면, html 따로 ,자바스크립트 따로(연결, 관리가 어려움)

자바스크립트 생성 방법

- 자바스크립트 함수라고 볼 수 있다
- 함수의 제목이 **대문자로**, 랜더링할 내용이 반드시 있을 것(JSX 코드)
    - 리액트 내장 컴포넌트와 커스텀 컴포넌트의 구분위해 대문자
        - 리액트 내장 컴포넌트 - > DOM으로 해석
        - 커스텀 컴포넌트 → 함수로 해석

```
function App() {
  return (
    <div>
      <header>
        <img src="assets/react-core-concepts.png" alt="Stylized
        <h1>React Essentials</h1>
        <p>
          Fundamental React concepts you will need for almost ar
          going to build!
        </p>
      </header>

      <main>
        <h2>Time to get started!</h2>
```

```
      </main>
    </div>
  );
}
```

```
<App><App /> 컴포넌트 사용
```

Javascript Syntax Extantion(JSX)

html + js 같이 쓸 수 있음 그러나 브라우저에서 실행이 안되는 코드(문법)임

리액트는 모든 컴포넌트에서 나온 jsx코드 결합하여 DOM을 생성

jsx의 목적은?

- 컴포넌트 내 타겟 HTML 코드를 정의할 수 있게 하기 위함
  - (직관, html과 유사한 구문으로 ui 구조 작성 가능)
- jsx 코드는 컴포넌트 트리를 불러옴
  - 리액트에게 각 컴포넌트 간 연관성을 알려줌
  - DOM 제어

```
index.js

import ReactDOM from "react-dom/client";
// 스크린에 앱 컴포넌트 랜더링

import App from "./App";
import "./index.css";

const entryPoint = document.getElementById("root");
ReactDOM.createRoot(entryPoint).render(<App />);
// 루트 컴포넌트 불러오는 메서드
```

동적 값 할당 { }

```
// 동적인 값 할당
const reactDescriptions = ["Fundamental", "Crucial", "Core"];

function genRandomInt(max) {
  return Math.floor(Math.random() * (max + 1));
}

function App() {
// 이 방식이 더 깔끔
    const description = reactDescipritions[genRandomInt(2)]
  return (
    <div>
      <header>
        <img src="assets/react-core-concepts.png" alt="Stylized
        <h1>React Essentials</h1>
        <p>
          {reactDescriptions[genRandomInt(2)]} React concepts yo
          almost any app you are going to build!
        </p>
      </header>

      <main>
        <h2>Time to get started!</h2>
      </main>
    </div>
  );
}

export default App;
```

동적 값 할당 2 : 이미지 불러오기

img 태그에 직접적 주소를 적기보다(src에)

```
import 이미지변수이름 from ‘assets/…’

img 태그 내 src 속성엔 { 변수명 }
```

컴포넌트 Props

- Props 란?
  - 컴포넌트에 (다른) 데이터를 전달하고 사용하기 위함
  - 컴포넌트의 재사용성을 높

```
function CoreConcept(props) {
  return (
    <li>
     <h2>{props.title}</h2>
      <img src="..." alt="" />
    </li>
  );
}


..

 <main>
        <section id="core-concepts">
          <ul>
            {/* Coreconcept 컴포넌트 사용 */}
```

```
            {/* 속성추가 가능함 */}
            <CoreConcept
              title="Components"
              description="the core Ui building block"
              images={변수명}
            />
          </ul>
        </section>
        <h2>Time to get started!</h2>
      </main>
```

```
=>
리액트가 객체(위 함수의 props)로 병합
{
    title: "Components",
    description: "the core Ui building block"
}

// 객체, 배열 등 다양한 dtype 전달 가
```

props 대체 표현법

```
ex. data.js import 해서 사용할 때

<CoreConcept
    title={CORE_CONCEPTS[0].title}
    description={CORE_CONCEPTS[0].description} />

위 대신
<CoreConcept {...CORE_CONCEPTS[0]} />
```

컴포넌트 생성 함수 대체 표현법(구조분해)

```
function CoreConcept({ image, title, description }) {
  return (
    <li>
     <h2>{title}</h2>
      <img src="..." alt="" />
    </li>
  );
}
```

rest property (...)

This pattern will store all remaining properties of the object or array into a new object or array.

```
const { a, ...others } = { a: 1, b: 2, c: 3 };
console.log(others); // { b: 2, c: 3 }

const [first, ...others2] = [1, 2, 3];
console.log(others2); // [2, 3]
```

```
예를 들어, 컴포넌트가 이런 식으로 사용되었다면:

<CoreConcept
  title={CORE_CONCEPTS[0].title}
  description={CORE_CONCEPTS[0].description}
  image={CORE_CONCEPTS[0].image} />
받은 prop들을 다음과 같이 단일 객체로 그룹화할 수 있습니다:

export default function CoreConcept({ ...concept }) {
  // ...concept groups multiple values into a single object
  // Use concept.title, concept.description etc.
  // Or destructure the concept object: const { title, descripti
```

기본 prop 값

type prop을 받는 커스텀 button 컴포넌트의 경우 type 속성도 같이 전달을 해줘야함

```
type 설정이 된 경우:

<Button type="submit" caption="My Button" />
되지 않은 경우:

<Button caption="My Button" />
이 컴포넌트가 작동하도록 하려면, type  Prop에 대한 기본 값을 설정할 수 있습
```

```
자바스크립트는 객체 비구조화(구조 분해 할당)를 사용할 때 기본 값을 지원함으로

export default function Button({ caption, type = "submit" }) {
  // caption has no default value, type has a default value of '
}
```

컴포넌트 사용을 위해서는 export를 해야한다

- 컴포넌트별 파일을 따로만든다
- 컴포넌트명과 동일할 것
- export, export default를 설정해준

컴포넌트별 css 분리

```
Header.css 파일을 만들면 된다

단, .js 파일에 css 파일 존재를 알려야함(import)
```

컴포넌트 관리

components 폴더 안 컴포넌트명 폴더 만들고, 컴포넌트명 파일들을 넣는다! 오예!

컴포넌트 “children” prop

- 열림과 닫힘 태그 사이에 있는 내용을 전달할 때 씀

컴포넌트 태그 사이 텍스트나 JSX 코드를 넣으면 리액트가 자동 무시

그럼 어떻게 하면 될까?

```
#app component

<Modal>
    <h2>Warning</h2>
    <p> do you want to delete? </p>
</Modal>


#modal component

function    Modal(props) {
    return <div id="modal">{props.children}</div>
    }
    // 즉 컴포넌트 태그와 태그 사이에는 props의 자식을 넣기 위한 곳이다
```

```
예시2

[방법3]

//이 방법 추천
#tabbutton.jsx
export default function TabButton(props) {
    return <li><button>{props.children}</button></li>

    #구조분해 할당도가능
```

```
    #tabbutton.jsx
export default function TabButton({children}) {
    return <li><button>{children}</button></li>

    # label 속성 지정으로도 가능
    <TabButton label='Component'></TabButton>
    export default function TabButton({label}) {
    return <li><button>{label}</button></li>


#app.jsx
import TabButton from ..

<TabButton> Components </TabButton> //여기선 텍스트
```

이벤트 처리하기

```
export default function TabButton({children}) {
    function handleClick() {
        console.log('Hello World!'
        }
    return (
        <li>
        // handleClick에 ()를 붙이지 말기! 클릭하면 실행되게 하기
        //<->  이 코드 라인 실행시 함수 실행됨
        // {} : jsx 내에서 javascript 표현식을 쓸때(함수 참조)
            <button onClick={handleClick}> {children} </button>
        </li>

참고로 <button>은 리액트 내장 컴포넌트
handleClick 함수는 내용을 동적으로 업데이트가 불가함(scoped)
```

⇒ 함수를 props로 전달하기 🔥

특정 버튼 클릭시 특정 content 보이게 하려면?

```
#TabButton.jsx
export default function TabButton({children, onSelect}) {

    return (
        <li>
            <button onClick={onSelect}> {children} </button>
        </li>



#app.jsx

function handleSelect() {
    console.log('Hello World! -- selected'
    }

<TabButton onSelect={HandleSelect}>Component</TabButton>
```

이벤트를 핸들링하는 함수의 실행을 다른 함수로 감싸면,

그 다른 함수가 이벤트 핸들링의 prop의 값으로 전달된다.

메인 함수(여기서는 handleClick)가 너무 빨리 실행되지 않게 한다.

이벤트가 발생할 때만 실행.

```
#TabButton.jsx
export default function TabButton({children, onSelect}) {

    return (
        <li>
            <button onClick={onSelect}> {children} </button>
        </li>
```

```
#app.jsx

function handleSelect(selectedButton) {
// selectedButton = 'a','b','c'
    console.log(selectedButton)
    }


        // handleselect 제어
        // 어떤 인자를 전달하는지 정의
<TabButton onSelect={ ()=> HandleSelect('a')}>Component</TabBut
```

컴포넌트는 한번만 실행된다

컴포넌트 함수가 재실행되지 않는다.

즉 ui 업데이트를 하려면? 재평가가 필요하다!

⇒ state 상태관리 & react hook 🔥

- == 업데이트
- 리액트에게 데이터가 변했다고 알려주는 것
- 변화가 생겼을 때 리액트가 컴포넌트를 재평가하게 하는 데이터

```
import { useState} from 'react'

function App() {

    useState() // hook 함수 // 업데이트 될 시 재평가하는 함수 // 다시 실

    // array일 것. 두개의 요소가 정의될 것.
```

```
    const [selectedTopic, setSelectedTopic] = useState('Please c


// 단 새로고침해보니 처음부터 다시 App component가 실행됨
// ui에는 상태가 업데이트되었지만 , 콘솔창에는 업데이트 이전의 값을 출력한다
```

=> 비동기 처리 !!

아래 코드는 올바르게 동작될까?

```
import { useState } from 'react';

function SomeComponent() {
  const [selected, setSelected] = useState(false);

  function handleClick() {
    const [selected, setSelected] = useState(true);
  }

  return <button onClick={handleClick}>Select</button>
}
```

→ 놉. 중첩 함수 내부에서 hook 함수를 호출x_x

데이터 기반 state 가져오기

```
import EXAMPLES .. //DATA

<div>{EXAMPLES[selectedTopic].title}</div>
```

조건적 콘텐츠 렌더링

```
{!selectedTopic ? <p> Please select </p> : null }
{!selectedTopic && (
    <div> ....

    )}
```

css 스타일링 및 동적 스타일링

```
JSX => class 대신 className 속성사용
<button className= {isSelected ? "active" : undefined}

// active는 따로정의한 css임


// isSelected 속성추가
<TabButton isSelected={selectedTopic === 'components'} // t/f 반
```

list 데이터 동적 출력🔥

- 같은 컴포넌트를 여러개 쓸 때(다른 데이터로)
  - map 메서드를 for문처럼 쓸 수 있음!

```
<ul>
    {배열.map((conceptItem) => <CoreConcept key={} { ...conceptI

</ul>
```