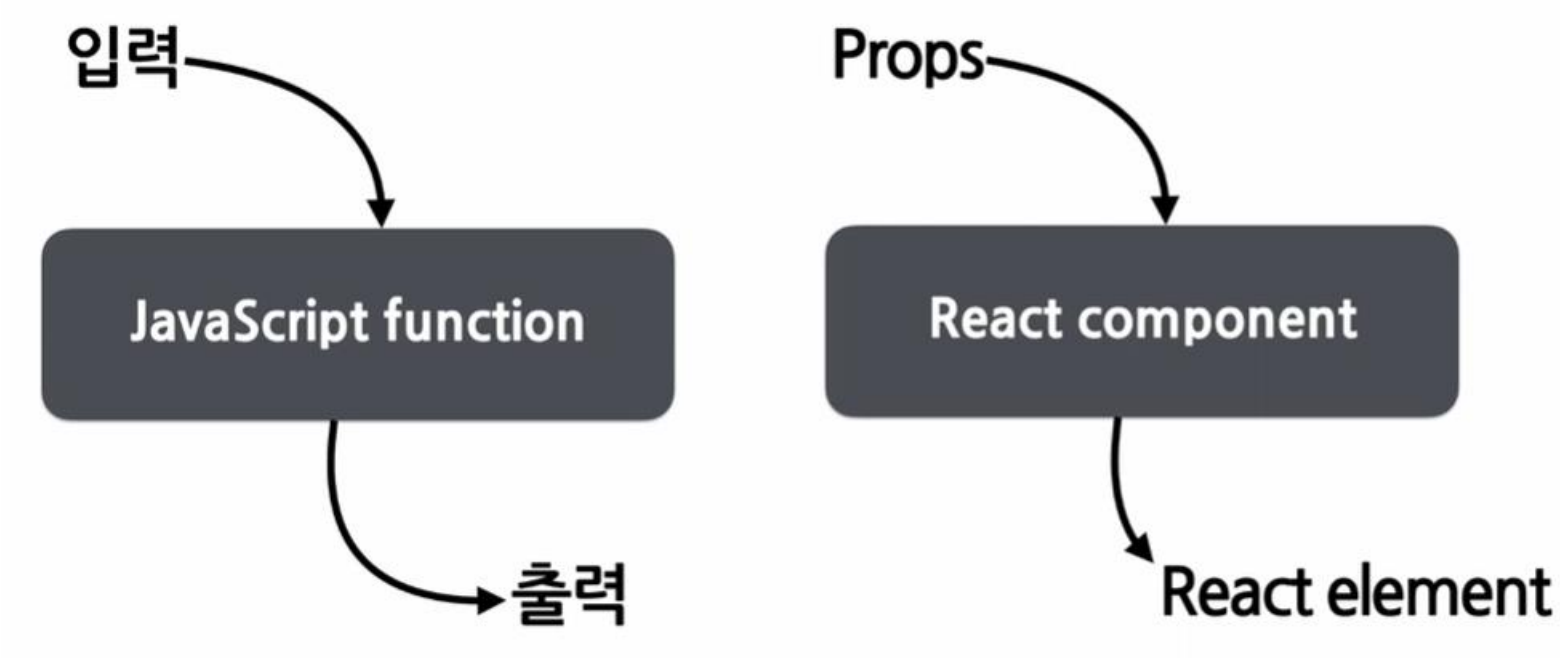
5. 컴포넌트와 Props

5.1 컴포넌트(Component)란?

- React는 컴포넌트 기반의 구조
  - 모든 페이지가 컴포넌트로 구성되어 있고, 하나의 컴포넌트는 또 다른 여러 컴포넌트들의 조합으로 만들 수 있음
  - 레고 블록 조립하듯 컴포넌트들을 모아서 개발
  - 반복되는 부분을 컴포넌트로 만들어 재사용
  - 결국 재사용 가능하도록 얼마나 컴포넌트를 잘 쪼개고 잘 조립하느냐가 React 개발

5.1 컴포넌트(Component)란?

- 자바스크립트 함수와 유사함
  - Props를 입력으로 받아서 그에 맞는 React 엘리먼트를 생성하여 반환



5.2 Props란?

- Property의 줄임으로 React 컴포넌트의 속성들을 의미
  - React 컴포넌트가 엘리먼트를 생성하기 위해 사용하는 값
  - 예시: 붕어빵 틀(컴포넌트) → 재료: 팥, 슈크림 등(Props) → 팥 붕어빵, 슈크림 붕어빵(엘리먼트)
  - 교재 p.145 그림 참고
  - 부모(상위) 컴포넌트가 자식(하위) 컴포넌트에게 값을 전달할 때 사용
- 컴포넌트에 전달할 다양한 정보를 담고있는 자바스크립트 객체

5.3 Props의 특징

- Read-Only
  - 읽기 전용 = 값을 변경 할 수 없다.
  - Props를 전달 받아 엘리먼트가 생성되는 도중에 변경되면 안됨
  - 다른 Props의 값으로 엘리먼트를 생성하려면..?
    - 새로운 값을 컴포넌트에 전달하여 새로 엘리먼트를 생성

순수 함수의 개념

- 입력값(input)을 바꾸려 하지 않고 항상 동일한 입력값에 대해 동일한 결과(output)를 반환 = ‘Pure’

```
function sum(a, b) {
  return a + b;
}
```

- 반면 다음 함수는 자신의 입력값(input)을 변경하기 때문에 순수 함수가 아님 = ‘Impure’

```
function withdraw(account, amount) {
  account.total -= amount;
}
```

5.3 Props의 특징

- **모든 React 컴포넌트는 자신의 props를 다룰 때 반드시 순수 함수처럼 동작해야 한다.**
- 모든 React 컴포넌트는 Props를 직접 바꿀 수 없고, 같은 Props에 대해서는 항상 같은 결과(엘리먼트)를 보여줄 것!
- 물론 애플리케이션 UI는 동적으로 변하기 때문에 뒤에서 배울 state를 통해 위 규칙을 위반하지 않고 사용자 액션이나 네트워크 응답 등으로 자신의 출력값(엘리먼트)을 변경 가능

5.3 Props의 사용법(실습)

- my-app/src/chapter5/5.3 실습
  - PropsUse.jsx
  - Profile.jsx
  - Layout.jsx
  - Header.jsx
  - Footer.jsx

5.4 컴포넌트 만들기

- 컴포넌트의 종류
  - **함수 컴포넌트와 클래스 컴포넌트**
- 클래스 컴포넌트
  - ES6의 클래스를 사용하여 만들어진 컴포넌트
  - React 초기 버전에서 주로 사용
  - 사용하기 불편함 → 함수 컴포넌트 + 훅(Hook)으로 대체
- 함수 컴포넌트(권장)
  - 자바스크립트 함수 형태로 된 컴포넌트
  - 코드가 간결해지고 사용하기 편함

5.4 컴포넌트 만들기

- 1) 함수 컴포넌트 VS 2) 클래스 컴포넌트
  - 1) Welcome이라는 이름의 함수가 props 객체를 전달받아서 인사말이 담긴 React 엘리먼트를 반환

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

  - 2) Welcome이라는 클래스

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

5.4 컴포넌트 만들기

- 컴포넌트의 이름 짓기
  - 항상 대문자로 시작해야 한다!
  - React는 소문자로 시작하는 컴포넌트를 HTML DOM Tag로 인식하려고 하기 때문
  - HTML div 태그로 인식

```
const element = <div />;
```

  - Welcome이라는 React 컴포넌트로 인식

```
const element = <Welcome name="Sara" />;
```

5.4 컴포넌트 만들기

- 컴포넌트 렌더링 - 페이지에 “Hello, Sara”를 렌더링하는 예시

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

const root = ReactDOM.createRoot(document.getElementById('root'));
const element = <Welcome name="Sara" />;
root.render(element);
```

  - React는 { name: ' Sara' }를 props로 하여 Welcome 컴포넌트를 호출
  - Welcome 컴포넌트는 <h1>Hello, Sara</h1> 엘리먼트를 반환
  - 엘리먼트를 인자값으로 root.render()를 호출
  - React DOM은 <h1>Hello, Sara</h1> 엘리먼트를 실제 DOM에 효율적으로 업데이트

5.5 컴포넌트 합성

- 여러 개의 컴포넌트를 합쳐서 하나의 컴포넌트를 만드는 것
- 컴포넌트 안에 또 다른 컴포넌트를 쓸 수 있음
- 복잡한 화면을 여러 개의 컴포넌트로 나눠서 구현 가능!

5.5 컴포넌트 합성

- 예시: Welcome 을 여러 번 렌더링하는 App 컴포넌트
  - App 컴포넌트는 3개의 Welcome 컴포넌트를 포함
  - 이렇게 여러 개의 컴포넌트를 합쳐서 또 다른 컴포넌트를 만드는 것을 컴포넌트 합성이라고 함

```jsx
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

function App() {
  return (
    <div>
      <Welcome name="Sara" />
      <Welcome name="Cahal" />
      <Welcome name="Edite" />
    </div>
  );
}
```

5.6 컴포넌트 추출

- 큰 컴포넌트에서 일부를 추출해서 새로운 컴포넌트를 만드는 것
- 복잡한 컴포넌트를 쪼개서 나누는 작업
- 컴포넌트를 어느 정도 수준까지 추출하는 것이 좋은지에 대해 정해진 기준은 없음
- 기능 단위로 구분하는 것이 좋고, 나중에 곧바로 재사용이 가능한 형태로 추출하는 것이 좋음
- 재사용 가능한 컴포넌트를 많이 갖고 있을수록 개발 속도가 빨라짐

5.6 컴포넌트 추출(실습)

- my-app/src/chapter5/5.6 실습
  - Comment.jsx
  - Avatar.jsx
  - UserInfo.jsx

5.7 댓글 컴포넌트 만들기(실습)

- my-app/src/chapter5 실습
  - Comment.jsx
  - CommentList.jsx
  - index.js