1. 리액트는 어쩌다 만들어졌을까?
   * Dom이 다양해지게 된다면, 이에 따라 업데이트를 하는 규칙도 많이 복잡해지기 때문에, 코드가 복잡해짐.
   * 리액트는 어떠한 상태가 바뀌었을 때, 그 상태에 따라 DOM을 어떻게 업데이트 할 지 규칙을 정하는 것이 아니라, 아예 다 날려버리고 처음부터 모든 걸 새로 만들어서 보여준다면 어떨까? 라는 아이디어에서 개발이 시작되었습니다.
   * 그러면 업데이트를 어떻게 해야 할지 에 대한 고민을 전혀 안해도 되기 때문에, 개발이 쉬워진다.
   * 동적인 UI를 보여주기 위해서 모든 걸 다 날려버리고 모든 걸 새로 만들게 된다면, 속도가 굉장히 느릴 것입니다.
   * 하지만, 리액트에서는 virtual dom 이라는 것을 사용해서 이를 가능케 했습니다.
   * Virtual dom 은 가상의 dom인데, 브라우저에 실제로 보여지는 dom이 아니라 그냥 메모리에 가상으로 존재하는 dom으로서 그냥 javascript의 객체이기 때문에 작동 성능이 실제로 브라우저에서 dom 을 본여주는 것보다 속도가 훨씬 빠릅니다.
   * 리액트는 상태가 업데이트 되면, 업데이트가 필요한 곳의 UI를 virtual Dom을 통해서 렌더링합니다.
   * 실제 브라우저에 보여지고 있는 DOM과 비교를 한 후, 차이가 있는 곳을 감지하여 이를 실제 DOM에 패치시켜줍니다.,
2. 작업환경 준비
   * Node.js : webpack과 bable 같은 도구들이 자바스크립트 런타임인 node.js를 기반으로 만들어져 있습니다.
     1. 자바스크립트 런타임 : 프로그래밍 언어가 구동되는 환경이다. 어떤 프로그래밍 언어가 동작할 수 있는 프로그램
   * -> 이전에는 자바스크립트 런타임이 브라우저 밖에 없었다. 하지만, 이제 node.js라는 새로운 자바스크립트 런타임이 나왔다.
   * Yarn : 조금 개선된 버전의 npm, node.js를 설치하게 될 떄 같이 딸려오는 패키지 매니저 도구. 프로젝트에서 사용되는 라이브러리를 설치하고 해당 라이브러리들의 버전 관리를 하게 될 때 사용. 우리가 yarn 을 사용하는 이유는 더 나은 속도, 더 나은 캐싱 시스템을 사용하기 위함.
   * Npm : node package manager로 일종의 앱스토어이다. 필요한 모듈을 다운로드 할 수 있는 모듈들이 모여있는 모듈스토어이다.
   * Package.json : 디펜던시 관리 파일이다. Package.json에는 이 프로그램을 실행시키기 위해 필요한 모듈들이 무엇인지, 프로그램을 실행시키는 방법, 프로그램을 테스트 하는 방법 등이 명시되어 있다.
   * Webpack, babel 은 무슨 용도?
     1. 리액트 프로젝트를 만들게 되면서 컴포넌트를 여러가지 파일로 분리해서 저장 할 것이고, 또 이 컴포넌트는 일반 자바스크립트가 아닌 JSX 라는 문법으로 작성하게 됩니다. 여러가지의 파일을 한개로 결합하기 위해서 우리는 Webpack이라는 도구를 사용하고, JSX를 비롯한 새로운 자바스크립트 문법들을 사용하기 위해서 우리는 babel이라는 도구를 사용합니다.
3. 나의 첫번째 리액트 컴포넌트
   * 리액트 컴포넌트를 만들 땐, import React from ‘react’를 통하여 리액트를 불러와주어야 합니다.
   * 리액트 컴포넌트에서는 xml 형식의 값을 반환해줄 수있는데, 이를 JSX라고 합니다.
     1. 코드의 최하단 ex) export default Hello;
   * 이렇게 해주면 다른 컴포넌트에서 불러와서 사용할 수 있습니다.
   * Index.js 에 있는 ReactDOM.render(<App/>, document.getElementById(‘root’));
     1. 여기서 ReactDOM.render의 역할은 브라우저에 있는 실제 DOM내부에 리액트 컴포넌트를 렌더링 하겠다는 것을 의미합니다.
     2. Public/index.html 을 열어보면, 내부에 <div id=”root”></div>
4. JSX의 기본 규칙 알아보기
   * <div> <b> Hello </b> world! </div>
   * -> javascript 로 변환하면, React.createElement(“div”, null, React.createElement(“b”, null, “Hello”,), “World”);
   * JSX는 리액트 컴포넌트 파일에서 XML 형태로 코드를 작성하면 babel이 jsx를 javascript로 변환해줍니다.
   * Babel은 자바스크립트의 문법을 확장해주는 도구 입니다. 아직 지원되지 않는 문법이나, 편의상 사용하거나 실험적인 자바스크립트 문법들을 정식 자바스크립트 형태로 변환해줌으로서 구형 브라우저 같은 환경에서도 제대로 실행 할 수 있게 해주는 역할을 합니다.
   * JSX가 Javascript 로 제대로 변환이 되려면 지켜주어야 하는 몇가지 규칙이 있다.
     1. 꼭 닫혀야 하는 태그
        1. HTML에서는 input 또는 br태그를 사용할 때 닫지 않고 사용하기도 함. 하지만 리액트에서는 그렇게 하면 안됨.
        2. 태그와 태그 사이에 내용이 들어가지 않을 때에는, Self Closing 태그라는 것을 사용해야합니다.
     2. 꼭 감싸져야 하는 태그
        1. 두개 이상의 태그는 무조건 하나의 태그로 감싸져 있어야 합니다.
        2. 단순히 감싸기 위하여 불필요한 div로 감싸는 게 별로 좋지 않은 상황이 있습니다. 그럴 땐, 리액트의 fragment라는 것을 사용하면 됩니다.
           1. -> fragment는 브라우저 사엥서 따로 별도의 엘리먼트로 나타나지 않습니다.
     3. JSX 안에 자바스크립트 값 사용하기
        1. JSX 내부에 자바스크립트 변수를 보여줘야 할 때에는 {}으로 감싸서 보여줍니다.
     4. Style과 className
        1. JSX에서 태그에 stye과 CSS class를 설정하는 방법은 HTML에서 설정하는 방법과 다릅니다.
        2. 인라인 스타일은 객체 형태로 작성을 해야 하며, background-color처럼 -로 구분되어 있는 이름들은 backgroundCOlor처럼 camelCase형태로 네이밍 해주어야 합니다.
           1. 인라인 스타일 : 한 줄짜리 짤막한 스타일 태그안에 직접 지정하여 사용.
        3. 그리고, CSS class를 설정 할 때에는 class=가 아닌 className=으로 설정을 해주어야 합니다.
        4. App.css : .gray-box{ background: grey; width:64px; height:64px }
        5. App.js => <div className=”gray-box”></div>
     5. 주석
        1. JSX 내부의 주석은 {/\* 이런 형태로\*/} 작성합니다.
           1. -> 중괄호로 감싸지 않으면 화면에 보입니다.
5. Props를 통해 컴포넌트에게 값 전달하기
   * Props는 properties의 줄임말입니다. 우리가 어떠한 값을 컴포넌트에게 전달해 줘야 할때, props를 사용합니다.
   * Props의 기본 사용법
     1. App컴포넌트에서 Hello 컴포넌트를 사용할 때 name 이라는 값을 전달하는 코드
     2. 여러개의 props, 비구조화 할당
     3. Function Hello({color, name}) {

<div style={{color}}>안녕하세요{name}</div>

}

* + 1. DefaultProps로 기본값 설정
       1. Hello.defaultProps = {
          1. Name: ‘이름없음’
       2. }
    2. Props.children
       1. 컴포넌트 태그 사이에 넣은 값을 조회하고 싶을 땐, props.children 을 조회하면 됩니다. (내부의 내용이 보여지게 하기 위해서는 Wrapper에서 props.children을 렌더링 해주어야합니다.
       2. Function Wrapper ({children}){
          1. Return(

<div style={style}>

{children}

</div>

)

}

Function app() {

<Wrapper>

<Hello/>

<Hello/>

</wrapper>

}

1. 조건부 렌더링
   * 특정 조건에 따라 다른 결과물을 렌더링 하는 것을 의미합니다.
   * Ex)
     1. Function Hello ({name, color, isSpecal}){
        1. Return(

<div style={{color}}>

{isSpecial && <b>\*</b>}

안녕하세요{name}

</div>

);

}

* + 참고로 JSX에서 null, false, undefined 를 렌더링하게 된다면 아무것도 나타나지 않게 된다.
  + Props 값 설정을 생략하면 ={true}

1. UseState를 통해 컴포넌트에서 바뀌는 값 관리하기
   * 컴포넌트에서 보여줘야 하는 내용이 사용자 인터랙션에 따라 바뀌어야 할 때 하는 구현 방법
   * OnClick={onIncrease()} 이렇게 하면 렌더링되는 시점에서 함수가 호출되버리기 때문에 이벤트를 설정할 때에는 함수 타입의 값을 넣어주어야 한다.
   * 동적인 값 끼얹기, useState
     1. 컴포넌트에서 동적인 값을 상태(state) 라고 부릅니다. 리액트에 useState라는 함수가 있는데, 이것을 사용하면 컴포넌트에서 상태를 관리할 수 있습니다.
     2. Const [number, setNumber] = useState(0);
        1. -> useState를 사용할 때에는 상태의 기본값을 파라미터로 넣어서 호출해줍니다. 이 함수를 호출해주면 배열이 반환되는데, 여기서 첫번째 원소는 현재 상태, 두번째 원소는 setter함수입니다.
        2. 원래 코드
           1. -> const numberState = useState(0);
           2. Const number = numberState[0];
           3. Const setnumber = numberState[1];
           4. 배열 비구조화 할당을 통하여 원소를 추출해주었다.
        3. 함수형 업데이트
           1. 기존 값을 어떻게 업데이트 할 지에 대한 함수를 등록하는 방식으로도 값을 업데이트 할 수 있습니다.
           2. Ex) setNumber(prevNumber => prevNumber – 1);
2. Input 상태 관리하기
   * Input에 입력하는 값이 하단에 나타나게 하고, 초기화 버튼을 누르면 input이 값이 비워지도록 구현하는 코드
   * InputSample.js
   * Ex)
   * Function InputSample() {
     1. Const [text, setText] = useState(‘’);
     2. Const onChange = (e) => {
        1. SetText(e.target.value);
        2. }
     3. Const onReset = () => {
        1. SetText(‘’);
     4. }
     5. Return (
        1. <div>
           1. <input onChange={onChange} value={text}/>
           2. <button onClick={onReset}>초기화</button>
           3. <b> 값 : {text} </b>

-> input의 상태를 관리할 때에는 input 태그의 value값도 설정해 주는 것이 중요.

그래야, 상태가 바뀌었을 때 input의 내용도 업데이트 된다.

1. 여러개의 input 상태 관리하기
   * Input에 name을 설정하고 이벤트가 발생 했을 떄 이 값을 참조하는 것이 가장 좋은 방법, 그리고 useState에서는 문자열이 아니라 객체형태의 상태를 관리해주어야한다.
   * Function InputSample() {
     1. Const [inputs, setInputs] = useState({
        1. Name : ‘’,
        2. Nickname:’’
     2. });
     3. Const { name, nickname } = inputs; //비구조화 할당을 통해 값 추출
     4. Const onChange = (e) => {
        1. Const { value, name } = e.target; // 우선 e.target에서 name과 value를 추출
        2. SetInputs({
           1. ...inputs, // 기존의 input객체를 복사한 뒤
           2. [name] : value //name키 를 가진 값을 value로 설정
        3. });
     5. });
     6. Const onReset = () => {
        1. SetInputs({
           1. Name:’’,
           2. Nickname:’’,
        2. })
     7. };
     8. Return (
        1. <div>
           1. <input name=”name” placeholder=”이름” onChange={onChange} value={name}/>
           2. <input name=”nickname” placeholder=”이름” onChange={onChange} value={nickname}/>
           3. <button onClick={onReset}>초기화</button>
           4. <div>

<b>값: </b>

{name} ({nickname})

* + - * 1. </div>
      1. </div>
      2. );
    1. }
  + 리액트 상태에서 객체를 수정해야 할 때에는 inputs[name] = value;
  + 이런식으로 직접 수정하면 안됩니다. 그 대신에, 새로우 객체를 만들어 새로운 객체에 변화를 주고, 이를 상태로 사용해주어야 합니다.