11. 2

Prettier – 코드를 이쁘게 해주는 아이. (텍스트를 일관성있게)

ESLint – 코드의 문법을 잡아주는 아이

**React 이벤트**

리액트에서 이벤트를 쓸 때는 인라인 이벤트 방법만 사용한다.

순수 자바스크립트에서는 카멜케이스로 이벤트를 선언하고 작성한다

이벤트는 DOM, 태그에서만 사용할 수 있다 컴포넌트에 쓰면 이벤트가 아니라 props로만 인식한다

이벤트는 반드시 태그에만 사용하자

onchange

class EventPractice extends Component {

  render() {

    return (

      <div>

        <h1> 테스트입니다. </h1>

        <input

          type="text"

          name="message"

          placeholder="아무거나 입력"

          onChange={(e) => {

            console.log(e.target.value);

          }}

        ></input>

      </div>

    );

  }

}

export default EventPractice;

class EventPractice extends Component {

  state = {

    message: '',

  };

  render() {

    return (

      <div>

        <h1> 테스트입니다. </h1>

        <input

          type="text"

          name="message"

          placeholder="아무거나 입력"

          value={this.state.message}

          onChange={(e) => {

            /\* state를 변경해주는 함수 \*/

            this.setState({ message: e.target.value });

            /\* state에서 선언한 message가 계속 바뀜

                  value도 지속적으로 변경 됨\*/

          }}

        ></input>

      </div>

    );

  }

}

        {/\* 온클릭 버튼 \*/}

        <button

          onClick={() => {

            alert(this.state.message);

            this.setState({ message: '' });

            /\* 온클릭 버튼 이후 setstate에 지정된 메세지가 나왔다가

            확인 버튼이 넘어가면 다시 메세지가 초기화 됨.

            state가 실시간으로 초기화 됨. 실시간으로 DOM이 업데이트 되는 특성 때문에 \*/

          }}

        >

          {

  /\* 이벤트 함수를 따로 빼는 방법 \*/

  /\* 1. constructor props 선언 \*/

  constructor(props) {

    super(props);

    this.handleChange = this.handleChange.bind(this);

    this.handleClick = this.handleClick.bind(this);

    /\* 3. bind라는 애를 이용하여 펑션들과 연결해준다 \*/

  }

  handleChange(e) {

    /\* 2. 이벤트 함수 실행될 때 되는 펑션만 바로 가져온다 \*/

    this.setState({ message: e.target.value });

  }

  handleClick(e) {

    alert(this.state.message);

    this.setState({ message: '' });

  }

  render() {

    return (

      <div>

        <h1> 테스트입니다. </h1>

        <input

          type="text"

          name="message"

          placeholder="아무거나 입력"

          value={this.state.message}

          /\* 4. 만든 애들을 각각 인터랙션 할 자리에 넣어준다. \*/

          onChange={this.handleChange}

        ></input>

        <button onClick={this.handleClick}> 확인 </button>

  /\* 조금 더 깔끔하게 하는 방법, 화살표 함수 이용\*/

  /\* const를 선언하지 않음. 작동은 똑같이 한다 \*/

  handleChange = (e) => {

    this.setState({ message: e.target.value });

  };

  handleClick = () => {

    alert(this.state.message);

    this.setState({ message: '' });

  };

/\* 2개 이벤트 운용 \*/

class EventPractice extends Component {

  state = {

    message: '',

    username: '',

    /\* 2. username 추가 \*/

  };

  handleChange = (e) => {

    this.setState({

      [e.target.name]: e.target.value,

    });

    /\* 5. 기존에는 메세지 값만 바뀌었는데, 나는 두개를 바꾸고 싶음. 어떻게 해야하나?

    message: e.target.value -> [e.target.name] : e.target.value

    알아서 이벤트가 발생되는 e.target에서, name값 (username, message) 만 가져옴 \*/

  };

  handleClick = () => {

    /\* 3. alert, setstate 수정 \*/

    alert(this.state.message + ':' + this.state.username);

    this.setState({ message: '', username: '' });

  };

  render() {

    return (

      <div>

        <h1> 테스트입니다. </h1>

        <input

          type="text"

          name="username"

          placeholder="아무거나 입력"

          value={this.state.username}

          onChange={this.handleChange}

          /\* 4. name, value 변경 \*/

        ></input>

        <input /\* 1. input 복사 \*/

          type="text"

          name="message"

          placeholder="아무거나 입력"

          value={this.state.message}

          onChange={this.handleChange}

        ></input>

  /\* 1. 키보드 관련 이벤트, 이벤트 객체 선언 \*/

  handleKeyPress = (e) => {

    if (e.key === 'Enter') {

      this.handleClick(); /\* 2. 엔터를 칠 경우 핸들클릭 작동되도록 선언 \*/

    }

  };

  render() {

    return (

      <div>

        <h1> 테스트입니다. </h1>

        <input

          type="text"

          name="username"

          placeholder="아무거나 입력"

          value={this.state.username}

          onChange={this.handleChange}

        ></input>

        <input

          type="text"

          name="message"

          placeholder="아무거나 입력"

          value={this.state.message}

          onChange={this.handleChange}

          onKeyPress={this.handleKeyPress} /\* 3. keyPress 추가 \*/

**functional component**

/\* functional Component \*/

import React, { useState } from 'react';

/\* 2. useState 가져옴 \*/

const EventPractice = () => {

  const [username, setUsername] =

    useState(''); /\* 1. useState 할 이름, 변경 변수 선언 \*/

  const [message, setMessage] = useState('');

  /\* 3. username, Message 내용 바꿔주는 함수 선언

     username의 state값을 바꾸는 역할을 함.\*/

  const onChangeUsername = (e) => {

    return setUsername(e.target.value);

  };

  const onChangeMessage = (e) => {

    return setMessage(e.target.value);

  };

  /\* 4. 클릭했을 때 username, message를 내보내줌 + 값 초기화 \*/

  const onClick = (e) => {

    alert(username + ' : ' + message);

    setUsername('');

    setMessage('');

  };

  /\* 5. 엔터값 지정 \*/

  const onKeyPress = (e) => {

    if (e.key === 'Enter') {

      onClick();

    }

  };

  return (

    <div>

      <h1> 테스트입니다. </h1>

      <input

        type="text"

        name="username"

        placeholder="아무거나 입력"

        /\* 6. 위에서 지정했던 대로 input 속성 수정 \*/

        value={username}

        onChange={onChangeUsername}

        onKeyPress={onKeyPress}

      ></input>

      <input

        type="text"

        name="message"

        placeholder="아무거나 입력"

        /\* 6.1 동일함 onChangeMessage, onKeyPress \*/

        value={message}

        onChange={onChangeMessage}

        onKeyPress={onKeyPress}

      ></input>

      {/\* 7. 확인에 onclick 넣어줌 \*/}

      <button onClick={onClick}> 확인 </button>

    </div>

  );

};

export default EventPractice;

////////////////////////////////실습 id, pwd받아서 공백인거 알림해주는거 이거 잘 안된다. 다시 해봐라!!!!!!!!!!!!!!

  /\* password의 값을 입력받은 값으로 바꿔줌 \*/

  handleChange = (e) => {

    this.setState({ password: e.tatget.value });

  };

  handleButtonClick = () => {

    this.setState({

      /\* 2. state값이 0000일경우 validated에 true가 적용됨. \*/

      clicked: true,

      validated: this.state.password === '0000',

    });

  };

  render() {

    return (

      <div>

        <input

          type="password"

          value={this.state.password} /\* 1. 위의 state에서 지정해줘야함 \*/

          onChange={this.handleChange}

          className={

            this.state.clicked

              ? this.state.validated

                ? 'success'

                : 'fail'

              : ''

          }

          /\* 버튼을 눌렀을 경우 +(validated 값이 트루냐 아니냐 인 여부) \*/

        ></input>

        <button onClick={this.handleButtonClick}> 검증하기 </button>

      </div>

    );

  }

}

**Ref -> id와 비슷한 역할을 함**

DOM을 직접적으로 건드려야할 때 쓰는 아이

1. 특정 input에 포커즈 줘야할 때

2. 스크롤 박스 조작할 때

3. canvas 요소 그림 그리기

          ref = {(ref) => {this.input=ref}}

    this.input.focus();

    /\* 이렇게 하면 input DOM 전체를 가져올 수 있음 \*/

const LiterationSample = () => {

  const names = ['눈사람', '얼음', '눈', '바람'];

  const nameList = names.map((name, index) => <li key={index}>{name} </li>);

  /\* key값을 인덱스 값으로 지정 \*/

  return <ul>{nameList}</ul>;

  /\* 데이터를 하나씩 뺴올 수 있는 map 함수 \*/

  /\* 리스트는 4개가 총 만들어진다. 배열형태로 담기게 된다. \*/

  /\* 배열을 컴포넌트로 뿌리고 싶은 경우, 고유값 key값이 있어야 한다 \*/

};

import React, { useState } from 'react';

const LiterationSample = () => {

  const [names, setNames] = useState([

    { id: 1, text: '눈사람' },

    { id: 2, text: '얼음' },

    { id: 3, text: '눈' },

    { id: 4, text: '바람' },

  ]);

  /\* setName는 names의 값을 바꿔주는 역할을 함 \*/

  /\* 객체데이터 배열 4개를 줘야함 \*/

  const [inputText, setInputText] = useState('');

  const [nextId, setNextId] = useState(5);

  /\* 입력된 값으로 업데이트 해줌 \*/

  const onChange = (e) => setInputText(e.target.value);

  const onClick = () => {

    const nextNames = names.concat({

      /\* 데이터 넣어주는 concat \*/

      id: nextId,

      text: inputText /\* names뒤에 계속 넣어주는 아이들\*/,

    });

  };

  setNextId(nextId + 1); /\* nextId를 하나씩 증가시켜줌 \*/

  setNames(nextNames);

  setInputText(''); /\* inputtext를 빈 문자열로 만들어줌 \*/

  const nameList = names.map((name) => <li key={name.id}>{name.text} </li>);

  return (

    <>

      <input value={inputText} onChange={onChange} />{' '}

      {/\* 이벤트 만들어 줘야함 const onChange \*/}

      {/\* inputtext의 값이 계속 바뀌게 됨  \*/}

      <button onClick={onClick}> 추가 </button>

      {/\* 버튼을 누를 때마다 리스트가 추가 되는 버튼 만들기 \*/}

      {/\* onClick 선언 \*/}

      <ul>{nameList}</ul>;

const LiterationSample = () => {

  const [names, setNames] = useState([

    { id: 1, text: '눈사람' },

    { id: 2, text: '얼음' },

    { id: 3, text: '눈' },

    { id: 4, text: '바람' },

  ]);

  const [inputText, setInputText] = useState('');

  const [nextId, setNextId] = useState(5);

  const onChange = (e) => setInputText(e.target.value);

  const onClick = () => {

    const nextNames = names.concat({

      id: nextId,

      text: inputText /\* names뒤에 계속 넣어주는 아이들\*/,

    });

    setNextId(nextId + 1); /\* nextId를 하나씩 증가시켜줌 \*/

    setNames(nextNames);

    setInputText(''); /\* inputtext를 빈 문자열로 만들어줌 \*/

  };

  const onRemove = (id) => {

    const nextNames = names.filter((name) => name.id !== id);

    setNames(nextNames);

  };

  const nameList = names.map((name) => (

    <li

      key={name.id}

      onDoubleClick={() => {

        onRemove(name.id);

      }}

      /\* 더블클릭시에 onRemove가 실행됨

      name.id가 없어진다. 그러면 onRemove = (id) 에 선택한 아이의 id가 들어가고(26번줄),

      최초에 선언한 애들과 들어간 id를 모두 찾고(5번줄), 숫자가 다른 아이들은 그래도 리턴시킨다(27번줄)

      그리고 리턴된 나머지 값은 그대로 setNames에 나오니 눈으로는 표가 남은 것으로 보인다 (28번줄,51번줄)

      \*/

    >

      {name.text}{' '}

    </li>

  ));

  return (

    <>

      <input value={inputText} onChange={onChange} />{' '}

      <button onClick={onClick}> 추가 </button>

      <ul>{nameList}</ul>

    </>

  );