React Part2

사용자 정보 입력받기
여러 입력 데이터 관리하기
selectbox 정보 입력받기
radio 정보 입력받기
checkbox 정보 입력받기
useRef hook 사용하기

React | 사용자 정보 입력받기

```jsx
function App() {
  let [data, setData] = useState('');

  return (
    <>
      <div>
        <input
          type="text"
          value={data}
          onChange={(e) => {
            setData(e.target.value);
          }}
        />
      </div>
      <p>입력받은 값 : {data}</p>
    </>
  )
}
```

- input 태그에 입력한 값을 저장하기 위한 state 변수를 생성한다.
- input태그의 값이 변경될때마다 실행되는 이벤트를 처리하는 onChange 이벤트 핸들러 속성을 추가한다.
- 이벤트 발생마다(input 태그의 값 변경이 일어날때마다) state변경함수를 호출하여 state변수의 값을 업데이트 한다.
- input태그에 입력된 값은 모든 이벤트정보가 담겨있는 이벤트 객체로부터 받을 수 있다.
- e.target.value는 이벤트가 발생한 태그의 value속성값을 의미한다.
- value 속성값은 input 태그에 입력된 데이터다.

React | 여러 입력 데이터 관리하기

```
function App() {
  let [info, setInfo] = useState({
    name : '',
    age : '',
    addr : ''
  });

  function changeInfo(e){
    setInfo({
      ...info,
      [e.target.name] : e.target.value
    });
  }

  return (
    <>
      <div>
        <input type="text" name='name' value={info.name}
          onChange={changeInfo} />
      </div>
      <p>입력받은 값 : {info.name} {info.age} {info.addr}</p>
    </>
  )
}
```

- 객체를 사용하여 입력받은 여러 정보를 한 번에 관리할 수 있다.
- 객체의 key값과 input 태그의 name 속성은 반드시 일치시킨다.
- state 변경함수의 인자에 spread 연산자를 사용하면 특정 데이터만 업데이트 할 수 있다.
- e.target.name은 이벤트가 발생한 태그의 name 속성값이다.
- e.target.value는 이벤트가 발생한 태그의 value 속성값이다.
- 객체의 key값을 변수로 사용 할 때는 []에 감싼다.

React | select box 정보 입력받기

```jsx
function App() {
  let [fruit, setFruit] = useState('orange');

  function changeFruit(e){
    setFruit(e.target.value);
  }

  return (
    <>
      <div>
        <select value={fruit} onChange={changeFruit}>
          <option value="apple">사과</option>
          <option value="banana">바나나</option>
          <option value="orange">오렌지</option>
        </select>
      </div>
    </>
  )
}
```

- select 태그에 onChange 이벤트 핸들러 속성을 추가한다.
- option 태그에는 반드시 value 속성을 추가한다.
- option을 선택하면 화면에 보이는 글자가 아닌 value 속성의 값이 데이터로 저장된다.
- value 속성이 없다면 option태그 사이에 작성한 글자가 데이터로 저장된다.
- select의 value 속성으로 지정한 값이 최초 화면 렌더링 시 선택되는 값이다.

React | radio 정보 입력받기

```
function App() {
  let [feel, setFeel] = useState('soso');

  function changeFeel(e){
    setFeel(e.target.value);
  }

  return (
    <>
      <div>
        <input type="radio" value={'sad'}
        onChange={changeFeel} checked={feel === 'sad'}/>슬픔
        <input type="radio" value={'soso'}
        onChange={changeFeel} checked={feel === 'soso'}/>그럭저럭
        <input type="radio" value={'happy'}
        onChange={changeFeel} checked={feel === 'happy'}/>기쁨
      </div>
    </>
  )
}
```

- state변수의 초기값은 radio 태그 중 화면에 먼저 보여질 radio 태그의 value 속성값으로 초기화한다.
- radio를 체크하면 눈에 보이는 글자가 아닌 value 속성의 값이 데이터다.
- checked 속성의 연산결과 true로 판명되는 radio를 체크한다.

* radio의 name 속성 값을 동일하게 작성하면 짝지가 됨

React | checkbox 정보 입력받기 - 1

자바스크립트 부분

```
//체크박스를 그릴 데이터
const fruitList = ['사과', '바나나', '오렌지'];
//체크하여 선택된 데이터가 담길 state 변수
const [checkedItems, setCheckedItems] = useState([]);

//전체 체크박스 클릭 시 실행 함수
function checkAll(e){
  setCheckedItems(e.target.checked ? fruitList : []);
}

//각각의 체크박스 클릭 시 실행 함수
function checkItem(e){
  if(e.target.checked){
    setCheckedItems([...checkedItems, e.target.value]);
  }
  else{
    //배열의 filter 함수는 return에 작성한 조건에 맞는 데이터만 걸러낸다
    const copyItems = checkedItems.filter((item) => {return item !== e.target.value});
    setCheckedItems(copyItems);
  }
}
```

React | checkbox 정보 입력받기 - 2

전체 체크박스

```
<input type="checkbox" onChange={(e) => {
  checkAll(e);
}}/>
```

각 행의 체크박스

```
<tbody>
  {
    fruitList.map((item, index) => {
      return (
        <tr key={index}>
          <td>
            <input type="checkbox" value={item}
              checked={checkedItems.includes(item)}
              onChange={(e) => {checkItem(e)}}/>
          </td>
          <td>{item}</td>
        </tr>
      )
    })
  }
</tbody>
```

React | useRef - 1

```
const [data1, setData1] = useState(1);
const [data2, setData2] = useState([1,2,3]);
const [data3, setData3] = useState({
  name : 'kim',
  age : 20
});

const data4 = useRef(1);
const data5 = useRef([1,2,3]);
const data6 = useRef({
  name : 'kim',
  age : 20
});

console.log(data4);
console.log(data5);
console.log(data6);
```

```
▶ {current: 1}
▶ {current: Array(3)}
▶ {current: {…}}
```

- useRef hook을 사용하면 useState와 마찬가지로 데이터를 저장할 수 있는 변수를 만들 수 있다.
- useState로 만들어진 변수와의 차이점은 값이 변경되도 component가 리렌더링되지 않는다는 점이다.
- useRef로 만들어진 변수는 객체 형태로 저장되며, key 값으로 current를 갖는다.
- 기본자료형의 변수와 useRef로 선언한 변수의 차이점의 리렌더링 시 값 초기화 여부이다. useRef로 선언한 변수는 리렌더링되어도 값이 초기화되지 않는다.
- 그렇기 때문에 component가 리렌더링되더라도 값을 지속적으로 관리하고 싶은 데이터는 useRef를 사용하여 선언한다.

React | useRef - 2

```jsx
function App() {
  const input1_tag = useRef();
  const [data, setData] = useState({
    tag_type : '',
    tag_id :'',
    tag_className : '',
    tag_value : ''
  });

  return (
    <>
      <input type="text" ref={input1_tag} id='aaa' className='bbb' />

      <button type='button' onClick={(e) => {
        setData({
          tag_type : input1_tag.current.type,
          tag_id :input1_tag.current.id,
          tag_className : input1_tag.current.className,
          tag_value : input1_tag.current.value
        });
      }}>input 태그 값 읽기!</button>

      <div>
        input 태그의 type 속성값 : {data.tag_type} <br />
        input 태그의 id 속성값 : {data.tag_id} <br />
        input 태그의 className 속성값 : {data.tag_className} <br />
        input 태그의 value 속성값 : {data.tag_value}
      </div>
    </>
  )
}
```

- useRef를 사용하면 다른 태그의 값을 참고할 수 있다.
- useRef로 변수 선언 시 초기값을 주지 않고, 값을 참조할 태그의 ref 속성으로 useRef로 선언한 변수를 넣어준다.
- 이렇게 하면 useRef로 선언한 변수는 태그 자체를 값으로 갖는다.
- 참조하는 태그의 모든 속성값은 객체의 데이터를 읽는 문법과 동일하다.
- input 태그에 입력한 내용은 value 속성값으로 접근할 수 있다.