React Part1

component
jsx
props
Event Handling
useState
map() 함수를 이용한 html 작성

```
function App() {
  return (
    <h2>안녕 리액트</h2>
  )
}
```

함수의 리턴문에서 html 코드를 반환하는 함수를 컴포넌트(component)라고 한다.

컴포넌트의 이름의 첫 글자는 반드시 대문자로 작성해야 리액트가 함수가 아닌 컴포넌트로 인식한다.

React는 이러한 컴포넌트를 만들고 조립하여 화면을 구성한다.

클래스를 이용하여 컴포넌트를 만들수는 있으나, 함수를 이용한 컴포넌트 생성을 권장한다.

작성한 컴포넌트는 App 컴포넌트 안에서 사용해야 화면에 반영된다.

컴포넌트 사용문법은 <컴포넌트명></컴포넌트명> 또는 <컴포넌트명 />이다.

각 컴포넌트는 부모 컴포넌트, 자식 컴포넌트의 관계를 가진다.

컴포넌트는 다른 jsx파일에 분리해서 만드는 것이 일반적이다.

컴포넌트 생성을 위한 파일명과 컴포넌트명은 동일하게 작성하며 확장자는 .jsx 이다.

'export default 컴포넌트명' 명령어는 외부 파일에서 컴포넌트를 사용할 수 있도록 내보내는 명령어이다.

rafce 에밋으로 간단하게 컴포넌트의 기본 틀을 만들 수 있다.

```jsx
import React from 'react'

const Header = () => {
  return (
    <h1>Header</h1>
  )
}

export default Header
```

Header.jsx 파일에 생성한 Header 컴포넌트

JSX는 JavaScript Extensions의 줄임말로 확장된 자바스크립트 문법을 의미한다.

순수 자바스크립트 문법에서는 리턴으로 html 코드를 반환할 수 없지만 jsx가 이러한 기능을 가능하게 한다.

jsx는 자바스크립트와 html을 혼용하여 사용하기에 자바스크립트의 변수를 html에 적용하여 동적 처리가 가능하다.

자바스크립트의 문법을 html에 적용 시 문자나 숫자와 같은 하나의 값으로 평가되는 코드만 가능하다. (if, for 불가능)

```
function App() {
  const title = '안녕 리액트';

  return (
    <>
      <Header/>
      <h2>{title}</h2>
    </>
  )
}
```

html에 변수 적용

boolean, null은 표현식에 사용해도 렌더링되지 않는다.

객체는 표현식에 사용하면 오류가 발생한다.

최상위 태그는 반드시 하나여야 한다.

모든 태그는 닫혀 있어야 한다.

html의 class 속성은 className으로 사용한다.

실습! 조건에 따른 다른 UI 표현해보기~

리액트의 부모 컴포넌트는 자식 컴포넌트에게 필요한 데이터를 전달할 수 있다.(부모 컴포넌트가 자식 컴포넌트에게만 전달 가능!)

이러한 데이터 전달에 사용하는 기능을 props라고 한다.

props 사용은 함수의 매개변수에 인자를 전달하는 것과 비슷한 내용으로 생각하면 된다.

```
const Footer = () => {
  return (
    <div>
      <Button title={'버튼1'} color={'red'}/>
      <Button title={'버튼2'} color={'blue'}/>
      <Button title={'버튼3'}/>
    </div>
  )
}
```

```
const Button = (props) => {
  return (
    <button style={{color:props.color}}>{props.title}</button>
  )
}

Button.defaultProps ={
  color : 'black'
}
```

props로 전달되는 데이터는 객체타입으로 전달된다.

자바스크립트의 구조분해할당 문법을 사용하면 props를 좀 더 편하게 받아 사용할 수 있다.

props로 전달되는 데이터가 변경되면 props를 전달받은 컴포넌트가 리렌더링된다.

DOM에 일어나는 여러 이벤트를 처리하는 것을 Event Handling 이라 한다.

이벤트란 클릭, 키입력, 창크기조절, 마우스호버 등 사용자가 ui와 상호작용하는 모든 것을 의미한다.

모든 이벤트는 태그의 속성으로 지정할 수 있으며 onClick, onChange 등과 같이 on으로 시작하는 대부분이 속성이 이벤트 속성이다.

Event를 handling 한다는 것은 이러한 이벤트가 일어났을 때 실행할 코드를 작성한다는 것이다.

ex> 버튼을 클릭했을 때 '클릭'이라는 문자열을 출력해주세요.

이벤트 핸들을 위한 함수에는 이벤트 객체가 매개변수로 전달되며, 이벤트 객체에는 이벤트에 대한 각종 정보가 들어있다.

```jsx
const Header = () => {
  return (
    <>
      <h1>Header</h1>
      <button type='button' onClick={(e) => {
        console.log('큭릭 이벤트 발생!');
      }}>버튼</button>
    </>
  )
}
```

```jsx
const Header = () => {
  function click(e){
    console.log('클릭 이벤트 발생!');
    console.log(e);
  }

  return (
    <>
      <h1>Header</h1>
      <button type='button' onClick={click}>버튼</button>
    </>
  )
}
```

함수의 이름만 전달!

다음과 같이 코드를 작성하면 버튼 클릭 시 바뀐 제목이 화면이 나타날까?

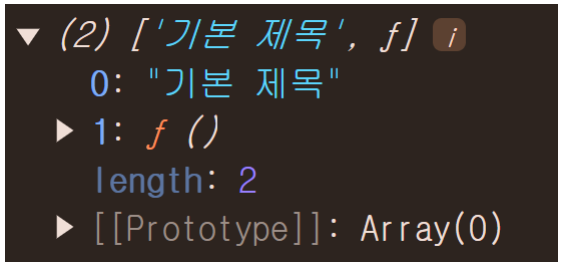
```
function App() {
  let title = '안녕 리액트';

  return (
    <>
      <Header/>
      <h2>{title}</h2>
      <button type='button' onClick={(e) => {
        title = '제목 수정';
      }}>제목변경</button>
      <Footer/>
    </>
  )
}
```

자바스크립트 문법으로 선언된 변수는 값이 변경되었다고 해도 변경된 정보가 화면에 반영되지 않는다.

리액트에서 제공하는 useState hook을 사용하여 선언한 변수만이 화면에 반영된다.

```
let title = useState('기본 제목');
console.log(title);
```



useState()의 소괄호 안에는 변수의 초기값을 작성한다.

useState()는 실행 후 2개의 데이터가 배열 형태로 반환된다.

배열의 첫번째 데이터는 useState()함수 안에 작성한 변수의 초기값이며, 두번째 데이터는 해당 변수의 값을 변경하는 함수다.

배열의 두번째 데이터로 반환되는 함수를 'state 변경 함수 ' 라 하며, state 변경 함수를 사용하여 변수의 값을 변경해야 변경 값이 화면에 반영된다.

```
let [title, setTitle] = useState('기본 제목');
```

useState()의 실행 결과 리턴되는 배열은 구조분해할당 문법을 사용하여 전달받는다.

state 변경 함수가 실행되면 state 변수가 선언된 컴포넌트가 리렌더링된다. 리덴더링이란 컴포넌트를 다시 읽어 화면을 새로 그린다는 의미이다.

리렌더링이 되더라도 state변수의 값은 초기화되지 않고, 마지막 데이터를 저장하고 있다.

아래 코드는 useState()를 사용한 코드다

```
function App() {
  let [title, setTitle] = useState('기본 제목');

  return (
    <>
      <Header/>
      <h2>{title}</h2>
      <button type='button' onClick={(e) => {
        setTitle('변경된 제목');
      }}>제목변경</button>
      <Footer/>
    </>
  )
}
```

```
function App() {
  let [title, setTitle] = useState('기본 제목');

  const changeTitle = (e) => {
    setTitle('변경된 제목');
  };

  return (
    <>
      <Header/>
      <h2>{title}</h2>
      <button type='button' onClick={changeTitle}>제목변경</button>
      <Footer/>
    </>
  )
}
```

결론! 값의 변화가 화면에 반영되어야 하는 변수는 useState() hook을 사용하여 선언하고, 반드시 state변경함수를 사용하여 값을 변경한다.

- useState의 초기값은 모든 자료형(숫자, 문자, 배열, 객체)의 데이터를 모두 사용할 수 있다.

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

- useState 변수는 모두 const로 선언한다.

  1) useState를 사용하여 만든 모든 변수는 내부적으로 참조 자료형으로 만들어진다(기본자료형도 마찬가지)

  2) 참조자료형으로 만들어지기 때문에 state변수는 값이 아닌 주소값을 가지고 있다.

  3) const로 선언하게 되면 참조값을 강제로 변경하는 코드 작성을 강제로 막을 수 있는 효과도 있다.

- state 변경함수는 state의 값(주소값)이 변경 될때만 리렌더링된다. 그래서 배열과 객체를 state 변경함수로 변경할 때 주의가 필요하다!
- useState로 선언된 변수의 값이 변경되면 화면이 리렌더링되기 때문에 화면에 변화가 필요한 경우 useState를 사용한다.
- 화면에 변화를 주고 싶다면 '어떻게 화면에 변화를 주지?'를 생각하는게 아니라, 'useState로 어떤 변수를 만들고, 변수의 값을 언제 변경하지'를 생각해야 한다.

state변수로 배열 데이터를 가질 때 변경코드

- 핵심은 배열은 주소값을 가지고 있기 때문에 값 변경을 위해서는 state변경함수의 인자로 새로운 배열을 전달 한다는 것이다!!!

```
function App() {
  const [arr, setArr] = useState([1, 2, 3]);

  return (
    <>
      {/* 버튼 클릭 시 배열 값 변경 이벤트 발생 */}
      <button type='button' onClick={(e) => {
        //구조분해할당 문법 사용으로 arr 배열과 같은 값을 갖는 새 배열 생성
        //새로 만들어진 배열은 기존 arr 배열과 같은 배열 데이터를 지니지만 주소값은 달라짐
        const copyArr = [...arr];
        copyArr[0] = 5;
        //state변경함수의 인자로 기존 arr과 다른 주소값을 지닌 배열을 전달하면
        //값 변경을 인지하여 리렌더링이 발생함!
        setArr(copyArr);
      }}>버튼</button>
    </>
  )
}
```

state변수로 객체 데이터를 가질 때 변경코드

- 핵심은 객체는 주소값을 가지고 있기 때문에 값 변경을 위해서는 state변경함수의 인자로 새로운 객체를 전달 한다는 것이다!!!

```
function App() {
  const [data, setData] = useState({
    name : 'Kim',
    age : 20,
    addr : '울산시'
  });

  return (
    <>
      {/* 버튼 클릭 시 객체 값 변경 이벤트 발생 */}
      <button type='button' onClick={(e) => {
        //구조분해할당 문법 사용으로 data 객체와 같은 값을 갖는 새 객체 생성
        //새로 만들어진 객체는 기존 data 객체와 같은 객체 데이터를 지니지만 주소값은 달라짐
        const copyData = {...data};
        copyData.name = 'Lee';
        //state변경함수의 인자로 기존 data와 다른 주소값을 지닌 객체를 전달하면
        //값 변경을 인지하여 리렌더링이 발생함!
        setData(copyData);
      }}>버튼</button>
    </>
  )
}
```

```
{/* 버튼 클릭 시 객체 값 변경 이벤트 발생 */}
<button type='button' onClick={(e) => {
  setData({
    ...data,
    name : 'Lee'
  });
}}>버튼</button>
```

객체는 key값이 중복일 때 가장 나중에 작성된 값만 가진다는 특징을 사용하면 위와 같이 좀 더 간단하게 코드를 작성할 수 있다.

실습 1> 클릭 횟수를 화면에 나타내는 코드 작성

클릭

실습 2> 클릭할 때마다 ON/OFF 로 바뀌는 버튼과 글자

ON OFF

OFF ON

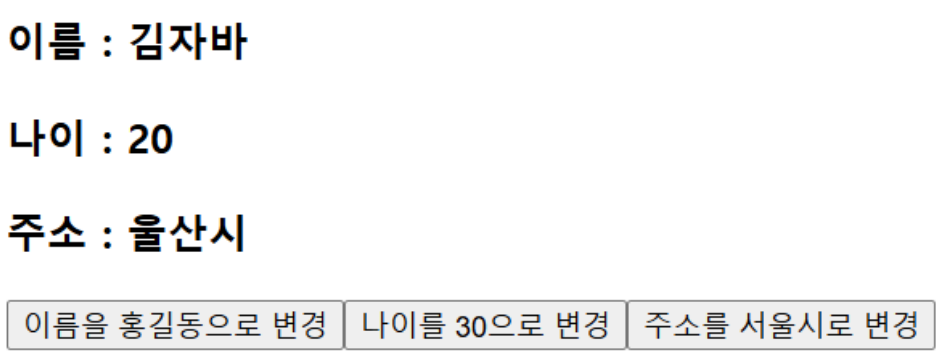
실습 3> 버튼 클릭으로 광고 보기/숨기기

광고닫기

광고보기

실습 4> 숫자를 클릭하면 클릭한 숫자만 1씩 증가(세 수는 하나의 배열을 사용할 것)

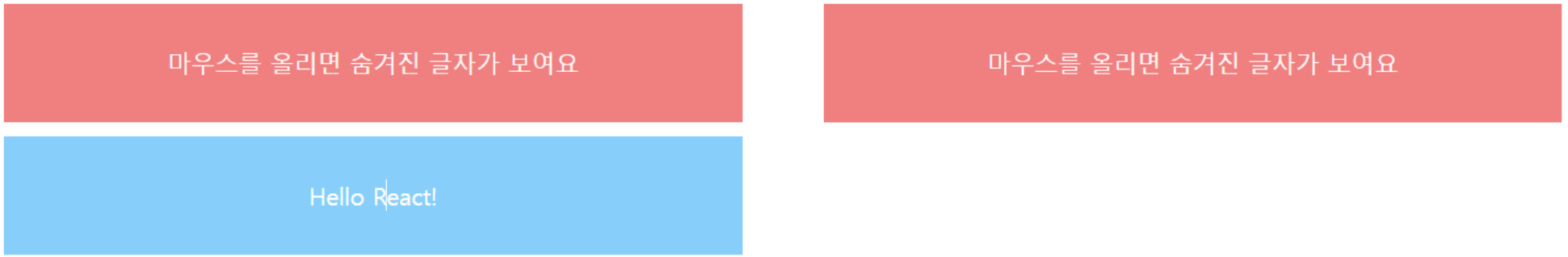
실습 5> 버튼 클릭 시 정보 변경(이름, 나이, 주소 정보는 하나의 객체를 사용하여 표한할 것)



실습6> -1, -10, -100, +100, +10, +1 각 버튼 클릭을 클릭하면 현재 카운트의 값을 변경하는 코드를 작성하세요.

실습6>마우스를 올리면 파란네모가 보이고, 마우스를 내리면 파란네모가 사라지게 해보세요.

태그에 마우스가 들어오면 발생하는 이벤트 속성 : onMouseEnter, 태그에서 마우스가 나가면 발생하는 이벤트 속성 : onMouseLeave



자바스크립트 배열의 forEach() 함수와 map() 함수

```
배열.forEach((배열 각 요소를 지칭할 변수, 배열의 인덱스) => {
   반복내용…
});

const arr = [1, 3, 5];
arr.forEach((item, index) => {
  console.log(`item = ${item}, index = ${index}`);
});
```

-결과-

item = 1, index = 0

item = 3, index = 1

item = 5, index = 2

```
배열.map((배열 각 요소를 지칭할 변수, 배열의 인덱스) => {
   반복내용…
   return 데이터;
});

const arr = [1, 3, 5];
const result = arr.map((item, index) => {
  console.log(`item = ${item}, index = ${index}`);
  return item * 2;
});

console.log(`result = ${result}`);
```

-결과-

item = 1, index = 0

item = 3, index = 1

item = 5, index = 2

result = [2, 6, 10]

```
function App() {
  let arr = [1,2,3];

  return (
    <>
      <ul>
        {
          arr.map((item, index) => {
            return (
              <li key={index}>{item}</li>
            )
          })
        }
      </ul>
    </>
  )
}
```

- 자바스크립트 배열의 forEach() 함수는 기본 for() 문법과 마찬가지로 반복 수행만 하지만 map() 함수는 반복 작업의 결과 값을 리턴해준다.
- 리액트 컴포넌트 안에서는 return되는 html을 화면에 뿌려주기 때문에 map() 함수의 리턴을 사용하면 반복되는 html을 그려줄 수 있다.
- map()을 이용해 반복적으로 그려지는 태그는 key속성에 태그의 구분값을 지정해줘야 한다.
- key속성의 값은 반복되어 그려지는 태그 각각이 유일하게 갖는 값이라면 무엇이든 가능하다.