0405 리액트 정리

◆ 자바스크립트

◎ import/export default

*기존 html에서 작성할 때 css와 js파일을 html에 연결해서 바로 사용

*react에서 js(jsx) 파일 중에 원하는 파일을 따로 불러와서 사용해야 한다.

1. import 하지않아도 export된 컴포넌트를 사용할수 있다 (O, **X**)
2. export default만 이용하면 하나만 내보낼 수 있다 (**O**, X)

*export default는 클래스/함수/변수 중 하나만 내보내서 사용할수 있다

다른 내용을 함께 내보내려면 export { 이름, 이름, 이름 } 과같이 여러개를 내보낼수 있다

◎ 화살표 함수

*익명함수를 더 짧게 쓰기위한 방법

*기본형태 : 함수이름 = (매개변수) => { 실행할 내용 }

*return을 적지않아도 return 값으로 전달하는 형태 : 함수이름 = (매개변수) => 리턴값

1. 아래 화살표 함수를 잘못 작성할 것을 고르세요

   addNum : 두 수를 받아와서 더한 값을 return 하는 함수

   1) const addNum = (a, b) => { return a+b }

   2) const addNum = (a,b) => a+b

   3) const addNum = (a,b) => { a+b }

   {}를 사용하고 return을 작성하지않으면 return값은 undefined로 값이 들어가지 않는다.

2. 위에 작성한 addNum(1,3) 의 return 값은 무엇이며 자료형은 무엇인지 적으세요

   1) 실행된 return의 결과값 : 4

   2) 실행된 return의 자료형 : 숫자형

3. return의 결과값이 객체 {} 일 경우 사용할 수 있는 함수 2개를 고르세요

   *자바스크립트는 함수내용을 감싸는 {}와 객체의 {}를 구분할 수 없다

   1) const objReturn = () => { return {id:1, name:"객체"}}

   2) const objReturn = () => { {id:1, name:"객체"} }

   3) const objReturn = () => ({id:1, name:"객체"})

   4) const objReturn = () => {id:1, name:"객체"}

◎ 배열 메소드 : 배열일 때 사용할 수 있다

1. map()

   https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/Array/map

   * 배열의 요소를 함수로 가져와서 사용, 함수 안에서 return 값으로 새로운 배열을 만듦

   const arrayMap**(새로운 배열)** = array.map( ()=>{ **return 배열에 들어갈 요소** } );

```
const array1 = [1, 4, 9, 16];

// Pass a function to map
const map1 = array1.map(x => x * 2);

console.log(map1);
// Expected output: Array [2, 8, 18, 32]
```

array.map( 함수 )
array.map( (array의 요소, array의 인덱스 )=>{
    array의 요소를 가져와서 사용 가능
    return 새로운 배열에 들어갈 값
})

array = [1,2,3,4] 가 있을 때 map()을 이용하여 아래의 배열을 작성하세요
1) arrayMap1 = [2,4,6,8] *각 요소에 2를 곱함
arrayMap1 = array.map( x => x*2 );

2) arrayMap2 = ["1", "2", "3", "4] *각 요소를 문자열로 변환
arrayMap2 = array.map( (x) => { return `${x}` } );

3) arrayMap3 = [1,4, 3, 8] *각 요소 중에서 짝수만 2를 곱함
```
arrayMap3 = array.map( (x) => {
    if(x %2===0 ) {
        return x*2;
    } else {
        return x;
    }
} );
```

arrayMap3 = array.map( x => x%2==0 ? x*2 : x ) 가능 *파란색은 화살표 함수

2. filter()
https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/Array/filter
* 배열의 요소를 함수로 가져와서 사용, 함수 안에서 return 값이 true이면 그 요소를 새로운 배열에 추가
const arrayFilter**(새로운 배열)** = array.filter( ()=>{ **return 조건식** } );

```
const words = ['spray', 'limit', 'elite', 'exuberant', 'destruction', 'present'];

const result = words.filter(word => word.length > 6);

console.log(result);
// Expected output: Array ["exuberant", "destruction", "present"]

```

array = [1,2,3,4,5,6,7] 가 있을 때 filter()을 이용하여 아래의 배열을 작성하세요
1) arrayFilter1 = [2,4,6] *짝수
arrayFilter1 = array.filter( x => x % === 0 )

2) arrayFilter2 = [1,2,3,4] *5보다 작은 수
arrayFilter2 = array.filter( (x) => { return x<5 } )

3) arrayFilter3 = [1,2,3,4,5,7] *6을 제외한 모든 수

arrayFilter3 = array.filter( (x) => x !== 6 )

arrayFilter3 = array.filter( (x) => { return x !== 6 } )

◎ 스프레드 연산자(...)

*객체나 배열 안에 있는 요소의 값들을 꺼내서 사용할 수 있다

객체 {id: 1, name:"홍길동", checked : true}

1) 객체의 값을 스프레드 연산자를 이용하여 id, checked 속성값은 동일하고 name 값을 "성춘향"으로 바꿔서 작성하세요

{...객체, name:"성춘향" }

2) 객체의 값을 스프레드 연산자를 이용하여 id, name 속성값은 동일하고 checked의 값을 부정(!) 해서 작성하세요

{...객체, checked : !객체.checked}

◎ 비구조 할당

*객체나 배열 안에 있는 요소들을 새로운 변수이름, 혹은 기존의 속성이름으로 사용할 수 있게 변수로 할당하는 방법

1) 객체 {text : "hi":, num:1} 값을 각각 비구조 할당을 하세요

const {text, num } = 객체

2) 배열 ["hi",1] 값을 각각 비구조 할당을 하세요

const [text, num] = 배열

◆ 리액트(JSX)

◎ 리액트 컴포넌트와 JSX (함수형 컴포넌트)

*리액트의 특징

① 가상 돔을 사용한다 (변수/state/props을 통해 가상돔을 만들어 제어)

② HTML안에 자바스크립트를 사용할 수 있다

③ 자주 사용하는 태그를 모아 컴포넌트로 만들어서 재사용할 수 있다

**1.** 아래 코드에서 HTML을 사용하는 공간을 고르세요 **(3)**

```
(1)
const ArrowTest = (props) => {
    const {name, check, children} = props
    (2)
    return (
        <div>   (3)
            { check && <h3>{name}</h3> }
            <h3>{check ? name : ""}</h3>
            <p>{children}</p>
        </div>
    );
}
export default ArrowTest;
```

2. HTML 안에서 자바스크립트를 작성하기 위해 {}를 사용하여 호출합니다

아래 코드중에서 HTML안에서 자바스크립트를 사용할 수 있게 하는 {}가 아닌 것을 고르세요 **(3)**

*map함수에서 사용하는 함수의 내용을 감싸는 괄호

```
return (
    <div>
        { check && <h3>{name}</h3> }
        <h3>{check ? name : ""}</h3>
        <p>{children}</p>
        {
            array.map((num)=>{
                return (<div>{num}</div>)
            })
        }
    </div>
);
```

(1) (2) (3)

◎ props

*부모 컴포넌트에서 자식 컴포넌트로 값을 전달할 때

1. 함수형 컴포넌트를 사용하여 아래 컴포넌트를 작성하세요

```
{/** props을 사용하는 클래스컴포넌트 작성*/}
<PropsComp color="blue">
  props값을 받아와서 글자색을 바꿉니다
</PropsComp>
```

```
PropsCompCopy = () => {
  const { color, children } = this.props;
  return <div style={{ color: color }}>{children}</div>;
};

export default PropsCompCopy;
```

◎ state

*각 컴포넌트 안에서 변경되는 값으로 저장되는 값

*함수형 컴포넌트에서는 useState()를 가져와서 사용한다.

1. 함수형 컴포넌트를 사용하여 아래 컴포넌트를 작성하세요

```
{/** props, state을 사용하는 클래스컴포넌트 작성
 * props의num값을 가져와서 버튼을 클릭할때마다 num씩증가
*/}
<CountPropsComp num={20} />
```

```jsx
import { useState } from "react";

CountPropsCompCopy = (props) => {
  const { num } = props;
  const [count, setCount] = useState(0);
  return (
    <div>
      <h4>{count}</h4>
      <button
        onClick={() => {
          setCount({ count: count + num });
        }}
      >
        +{num}
      </button>
    </div>
  );
};

export default CountPropsCompCopy;
```

◎ 이벤트와 메소드

*이벤트의 함수를 넣어줄 때, 안에 는 함수의 형태/메소드이름을 전달하여 사용해야한다

*함수이름() 작성하면 그 함수가 실행이되고 return 값이 남게되고 이벤트가 발생할때는 아무일도 일어나지 않는다.

onClick = {()=>{}}

◎ 배열과 반복되는 컴포넌트

*배열의 map을 이용하여 배열의 개수만큼 반복해서 화면에 출력할수 있다.

*map을 이용해서 만든 배열을 변수에 담아서 사용할수 도 있지만

다시사용할 일이 없다면 변수에 담지않고 {} 안에 바로 사용할수 도 있다

◎ 배열의 값 추가, 제거, 수정 코드 확인하는 순서

0. 배열의 값 화면에 출력 : map()을 이용하여 화면에 태그/컴포넌트로 감싸서 출력

```jsx
{students.map((student) => (
  <li key={student.id} className={student.checked ? "on" : ""}>
    <input
      type="checkbox"
      checked={student.checked}
      readOnly
      onClick={()=>{checkedStudent(student.id)}}
    />
    {student.id} , {student.name}
    <button
      onClick={() => {deleteStudent(student.id)}}
    >
      X
    </button>
  </li>
))}
```

1. 추가 : concat()을 이용하여 새로운 값이 추가된 배열 생성

*button을 Click했을 때 실행

```
<h3>학생추가</h3>
<input type="text" onChange={inputChange} value={inputName} />
<button onClick={addStudent}>추가</button>
```

```
const addStudent = () => {
  // 값을 받아와서 새로운 배열로 만들기
  // 새로운 배열 students 할당
  const newStudents = students.concat({
    id: globalId,
    name: inputName,
  });
  globalId++;
  setStudents(newStudents);
  setInputName("");
};
```

2. 제거 : filter()를 이용하여 특정값을 제외한 배열 생성

*button을 Click했을 때 실행

```
<button
  onClick={() => {deleteStudent(student.id)}}
>
  X
</button>
```

```
// id값을 전달하여 메소드 안에서 사용
const deleteStudent = (id) => {
  const newStudents = students.filter((s) => s.id !== id);
  setStudents(newStudents);
};
```

3. 수정 : map()을 이용하여 배열의값을 수정한 배열 생성

```
<input
  type="checkbox"
  checked={student.checked}
  readOnly
  onClick={()=>{checkedStudent(student.id)}}
/>
```

*checkbox를 Click했을 때 실행

```
const checkedStudent = (id) => {
  const newStudents = students.map((s) => {
    if (id !== s.id) {
      return s;
    } else {
      return { ...s, checked: !s.checked };
    }
  });
  setStudents(newStudents);
}
```