

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

addNum(1,3); => return값이 저장되지않고 사라짐

let result = addNum(1,3); => 함수의 return값을 변수에 저장해서 사용

console.log(addNum(1,3)); => 콘솔창에 값을 출력하고 사라짐

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"}) => return값으로 구분

소괄호의 사용범위 : if() / ()&&() / 1+3+(5-1)

=> 사용용도 1)구분 2)먼저 실행 할 연산

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

◎ 배열 메소드 : 배열일 때 사용할 수 있다

리액트의 특징 : 가상돔 > 데이터 - 가상돔(태그)을 얹어서 생각

! 변수의 자료형에 대해서 알고 사용해주어야 한다 !

그 중에서 여러 개의 데이터를 사용하는 배열

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 만들

`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}` });`

String(x) => 문자열로 반환

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 });`

```

1 const words = ['spray', 'limit', 'elite', 'exuberant', 'destruction', 'present'];
2
3 const result = words.filter(word => word.length > 6);
4
5 console.log(result);
6 // Expected output: Array ["exuberant", "destruction", "present"]
7

```

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) => ! (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 / const {num, text} = 객체

2) 배열 ["hi",1] 값을 각각 비구조 할당을 하세요 : ex) useState()

const [text, num] = 배열 => 이름은 원하는대로 적어줄 수 있지만, 순서는 고정

◆ 리액트(JSX)

◎ 리액트 컴포넌트와 JSX(함수형 컴포넌트)

* 리액트의 특징

- ① 가상 돔을 사용한다 (변수/state/props를 통해 가상돔을 만들어 제어)
- ② HTML안에 자바스크립트를 사용할 수 있다 : JSX
- ③ 자주 사용하는 태그를 모아 컴포넌트로 만들어서 재사용할 수 있다

1. 아래 코드에서 HTML을 사용하는 공간을 고르세요 (3)

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

(1) => import, 컴포넌트 안에서 전역으로 쓸 변수 (JS영역)

- * 변수 안에 원한다면 <p></p> 태그 사용가능
- 이때 이 태그는 지금 화면에 출력X, 변수에 할당O

(2) => 전역변수를 선언, 할당하는 공간 (JS영역)

- useState(); 변수, 메소드 사용(const, let을 통해 할당하여 사용)
- * 변수 안에 원한다면 태그 사용가능
- 선언, 할당하는 것으로 HTML이 화면에 출력되지않는다

* 사용하는 메소드의 종류 - 이벤트에 사용하는 함수 / return 화면 출력하는 함수

이벤트에 사용하는 함수 => 함수이름만 이용해서 호출 : addMemo

- * onClick / onChange와 같이 이벤트가 발생했을 때 실행하는 함수
- * return을 사용하지않음 - 안에 있는 내용 실행하는 것이 중요

return값을 이용한 화면 출력함수 : 함수이름과 ()를 통해서 호출 : printClock()

: HTML 화면에 작성할 { }안의 자바스크립트내용이 길어질 때,
함수도 만들어서 필요한 값만 return해서 HTML에 출력하기 위한 용도
return값 중요, return값이 화면에 출력되기에 그 값이 문자, 숫자, 배열(객체배열 제외)로
return해주어야한다

* 객체가 return되지않게 주의!

- * 변수 : state값을 수정해서 사용할 때 저장 또는 고정값을 저장
 - * 컴포넌트 안의 변수는 업데이트마다 초기화(고정값)
 - * 고정되는 값을 갖는 것이 싫다면 state의 값을 가져와서 사용

(3) => HTML로 작성한 화면을 브라우저에 출력하는 공간

: JSX를 사용하고 있기 때문에 html안에서 { } 통해 자바스크립트 작성!

* JSX의 특징

<p 속성={}> { } </p> 태그 안에서 { } 통해 자바스크립트 작성 가능

{ }의 역할 => 자바스크립트의 변수 값을 출력

삼항연산자를 이용해서 그 결과 출력

✓ { }에 작성된 내용의 출력될 값을 생각하고 작성

✓ 객체 자체를 출력하지않게 주의

가장 가까이에 있는 코드 확인

{ }의 용도

자바스크립트 - 함수{ }, for{ }, if{ } => 코드공간묶음

객체표기 { } => 자료형

JSX - HTML안에서 자바스크립트 쓰는 공간 => { }가 HTML안에 작성

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

◎ props

*부모 컴포넌트에서 자식 컴포넌트로 값을 전달할 때

1. 함수형 컴포넌트를 사용하여 아래 컴포넌트를 작성하세요

props로 전달할 수 있는 내용 : "문자열", { }를 통해서 숫자, 문자, 배열, 객체, 메소드,

children 공간을 통해서 값 가져옴 : 문자열, {}를 통해서 숫자, 문자, 배열객체, 컴포넌트 전달가능

{/* props를 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

`const [value, setValue] = useState(값)`

`setValue` 값을 수정하면 업데이트가 되면서 `value`값과 화면 수정

*`value` = 값 : 직접 할당해서 수정X - 화면 수정X

`useState`에 들어갈 수 있는 값 : 숫자, 문자, 변수, 배열, 객체, 메소드(`return`값)

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

```
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 `map`을 이용하여 배열의 개수만큼 반복해서 화면에 출력할수 있다.

: 화면에 출력 => 배열의 `map`은 자바스크립트의 내용 { }를 사용해서 `return`값을 출력

: 배열의 `map`의 `return` 값은 배열이기 때문에 변수 안에 담아서 사용가능

ex) `const arrayMap = array.map(()=>{함수 내용});`

return의 HTML 공간에서 { arrayMap }으로 사용가능

*map을 이용해서 만든 배열을 변수에 담아서 사용 할 수도 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

```
{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

리엑트는 가상돔을 사용해서 화면 수정

=> 데이터 값과 가상돔을 연결해서 사용

: inputName(state) = <input>의 value 값

: inputName(state) = onChange의 setInputName으로 값 수정

학생 이름을 받아올 공간

```
const [inputName, setInputName] = useState("");
```

<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("");
};
```

checked 값이 없어도 진행되는 이유,
undefined == false(boolean) 값과 동일

2. 제거 : filter()를 이용하여 특정값을 제외한 배열 생성

*button을 Click했을 때 실행

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

true => 값을 출력 / false => 값을 사용X

```
// 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);
}
```