0. 컴포넌트?

특정 코드 뭉치를 다른 부분에 이식하거나 재사용하기 위해 사용하는 코드 블록 단위

파일 단위로 작성한 후 필요한 위치에서 임포트해 사용할 수 있다.

1. component 사용원리?

첫 번째 줄에서 import한 {component}를 상속받아 R003 클래스(새로만든js파일)에서 사용

return된 html코드를 render()함수를 사용해 화면에 표시

App.js 에서 import해서 사용가능

2. 생명주기?

component의 생성, 변경, 소멸 과정

생성 : constructor() getDerivedStateFormProps() render() componentDidMount()

3. render()?

render()는 return되는 html형식의 코드를 화면에 그려주는 함수

화면 내용이 변경돼야 할 시점에 자동으로 호출

4. constructor(props)

생명주기 함수 중 가장 먼저 실행되며, 처음 한번만 호출

component 내부에서 사용되는 변수(state)를 선언하고

부모 객체에서 전달받은 변수(props)를 초기화할 때 사용한다.

super()함수는 가장 위에 호출해야 한다

5. getDerivedStateFromProps(props, state)

constructor() 다음으로 실행된다.

component가 새로운 props를 받게 됐을 때 state를 변경해준다.

App.js에서 전달한 prop\_value라는 변수를 props.prop\_value로 접근해 값을 가져올 수 있다.

6. componentDidMount()

작성한 함수들 중 가장 마지막으로 실행된다.

render()가 return 되는 html 형식의 코드를 화면에 그려진 후 실행된다.

화면이 그려진 후에 실행돼야하는 이벤트 처리, 초기화 등에서 가장 많이 활용되는 함수다.

7. setState()

변수의 선언과 초기화를 동시에 실행한다.

8. shouldComponentUpdate(props,state)

setState()로 인해서 state의 변경이 발생했기 떄문에 ‘변경’단계의 생명주기 함수 shouldComponentUpdate()가 실행된다.

Boolean 유형의 데이터를 반환하는데 return이 true일 경우 render()를 한 번 더 호출한다.

NOTE

함수의 반환값에 따라 render() 재실행 가능한다는 점을 이용

Props나 state변수가 변경될 때 화면을 다시 그리며 제어할 수 있다.

* 직접 해보니 componentDidMount까지 왔다가 constructor 건너뛰고 getDerived 나오고 render 나옴 새로운 getDerived 나오고부터 this.state가 변경됨

9. ES는 표준화된 스크립트 언어

ES 뒤에 붙은 숫자는 버전

파이썬에서의 f-string == js에서의 백틱+${변수}

f’hello {name}’ == `hello ${name}`

아래는 파이썬과 동일함

startswith

endswith

파이썬의object in list == includes

arr.some도 있음

Object not in list

10. ES5에서 사용하던 var는 변수를 재선언 재할당할 수 있다.

=>변수의 사용범위가 불확실해지거나 의도하지 않은 변숫값 변경 발생

ES6에서 단점을 보완하기 위해서 let const 추가

let은 재할당 허용

const 재선언 재할당 허용하지 않음

11~16

11. 전개 연산자 사용하기

배열이나 객체 변수를 직관적이고 편리하게 합치거나 추출할 수 있게 도와주는 문법

변수 앞에 …(마침표3개)

//js array

        var varArray1 = ['num1','num2'];

        var varArray2 = ['num3','num4'];

        var sumVarArr = [varArray1[0],varArray1[1],varArray2[0],varArray2[1]];

        // var sumVarArr = [].concat(varArray1,varArray2);

        console.log(`1. sumVarArr : ${sumVarArr}`);

        // ES6 Array

        let sumLetArr = [...varArray1,...varArray2];

        console.log('2. sumLetArr : '+sumLetArr);

        const [sum1,sum2,...remain] = sumLetArr;

        console.log('3. sum1 : '+sum1+', sum2 : '+sum2+', remain : '+remain);

        var varObj1 = {key1 : 'val1',key2:'val2'};

        var varObj2 = {key2:'new2',key3:'val3'};

        // javascript Object

        var sumVarObj = Object.assign({},varObj1,varObj2);

        console.log('4. sumVarObj : '+JSON.stringify(sumVarObj));

        // ES6 Object

        var sumLetObj = {...varObj1,...varObj2};

        console.log('5. sumLetObj : '+JSON.stringify(sumLetObj));

        var {key1,key3,...others} = sumLetObj;

        console.log('6. key1 : '+key1+', key3 : '+key3+', others : '+JSON.stringify(others));

12. class

ES5 prototype과 동일

JS는 함수를 객체로 사용할 수 있다

200제 51페이지 직접 보는게 좋을 듯

13. 화살표 함수

Function 대신 => 사용

This로 component의 state 변수에 접근해 사용할 수 있다.

Return 문자열 생략, this를 bind해야하는 문제도 생기지 않음

콜백함수내부에서는 this가 window객체이기 때문에 this로 state변수에 접근하려면 에러발생

.bind(this) 해주면 component 사용가능

14. forEach() 함수

배열의 처음부터 마지막순번까지 모두 작업하는 경우 forEach()문 쓰는 것이 간편

        ForEach\_Arr.forEach((result)=>{

            ForEach\_newArr.push(result);

        })

순번과 배열의 크기 정보를 사용하지 않음

0부터 배열의 크기만큼 반복하여 순서대로 배열 값 반환

반복문이 실행될 때마다 콜백 함수로 result를 받아 ForEach\_newArr에 넣는다

15. map()

forEach()와 마찬가지로 for문에서 사용하던 순번과 배열의 크기 변수를 사용하지 않음

차이점은 forEach()와 달리return을 사용해 반환값을 받을 수 있음

화살표 함수는 return 구문을 생략할 수 있다

Map\_Arr.map(x=>x) 동일Map\_Arr.map(x=>{return x})

파이썬 map이랑 유사한듯

16. jQuery

    input\_alert = (e) => {

        var input\_val = $('#inputId').val();

        alert(input\_val);

    }

        <input id = "inputId" name = "inputName"/>

        <button id = "buttonId" onClick={e => this.input\_alert(e)}>

         jQuery Button

        </button>

제이쿼리 접근할 때$붙여서 사용한다 id가 inputId인 <input>태그에 접근

input태그의 value값을 input\_val에 저장