**단축 평가 논리 계산법**

console.log(true && 'hello'); // hello

console.log(false && 'hello'); // false

console.log('hello' && 'bye'); // bye

// Truthy한 값이면 뒤에 값 표시

console.log(null && 'hello'); // null

console.log(undefined && 'hello'); // undefined

console.log('' && 'hello'); // ""

console.log(0 && 'hello'); // 0

console.log(1 && 'hello'); // hello

console.log(1 && 1); // 1

// &&

// 특정 값이 유효한 경우에만 어떤 값을 조회할 때 사용

const object = null;

const name = object && object.name;

console.log(name); // null

const object = { name : 'Jin' };

const name = object && object.name;

console.log(name); // Jin

// ||

// 어떤 값이 없을 때 다른 값을 사용할 때 사용

const namelessDog = {

name: '',

};

function getName(animal) {

const name = animal && animal.name;

return name || '이름이 없습니다';

}

const name = getName(namelessDog);

console.log(name); // 이름이 없습니다

단축 평가 논리 계산법은 리액트에서 조건부 랜더링할 때에 유용하게 사용할 수 있다.

# 비구조화 할당

const deepObject = {

state : {

information : {

name : 'Jiwon',

languages : ['Korean', 'English']

}

},

value: 5

}

// 비구조화 할당 두 번 하기

const { name, languages } = deepObject.state.information;

const { value } = deepObject;

const extracted = {

name,

languages,

value

};

console.log(extracted);

// 선호 되지 않는 방식

const {

state : {

information : {

name, languages

}

},

value

} = deepObject;

const extracted = {

name,

languages,

value

};

console.log(extracted);

# Class 클래스

### MDN 문서

<https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Classes>

### constructor

class A {}

console.log(new A());

// A{}

class B{

constructor(){

console.log('constructor');

}

}

console.log(new B());

// constructor

// B{}

class C {

constructor(name, age){

console.log('constructor', name, age);

}

}

console.log(new C('Mark',37));

// constructor Mark 37

// C{}

console.log(new C());

// constructor undefined undefined

// c{}

### 멤버변수 - 객체의 프로퍼티

// 1

class A {

constructor(name,age){

this.name=name;

this.age=age;

}

}

console.log(new A('Mark',37)); // A{name:'Mark', age:37}

// 2

class B {

name;

age;

}

console.log(new B()); // 런타임 오류 주의

// 3

class C {

name='no name';

age=0;

constructor (name, age){

this.name=name;

this.age=age;

}

}

console.log(new C('Mark',37));

// {name:'Mark', age:37}

### 멤버함수

class A {

hello1(){

console.log('hello1'.this);

}

hello2=()=>{

console.log('hello2'.this);

};

}

new A().hello1(); // hello1 A{hello2: [function: hello2]}

new A().hello2(); // hello2 A{hello2: [function: hello2]}

class B {

name = 'Mark';

hello(){

console.log('hello', this.name);

}

}

new B().hello(); // hello Mark

### get, set

// 외부에서 값을 바꾸지 않고

// 내부적으로 쓸 경우 \_언더바 사용

class A {

\_name='no name';

get name(){

return this.\_name + '@@@';

}

set name(value){

this.\_name=value + '!!!';

}

}

const a = new A();

console.log(a); // A{ \_name: 'no name' }

a.name = 'Mark'; // set 동작

console.log(a); // A{ \_name: 'Mark!!!' }

console.log(a.name); // Mark!!!@@@

console.log(a.\_name); // Mark!!!

### static 변수, 함수

class A {

static age = 37;

static hello(){

console.log(A.age);

}

}

console.log(A, A.age); // [function:A] {age:37} 37

A.hello(); // 37

class C {

static name = '이 클래스의 이름은 C가 아니다';

}

console.log(C);

// [Function: 이 클래스의 이름은 C가 아니다]

// { name: '이 클래스의 이름은 C가 아니다' }