10. 25

**어제 했던 레지스터 폼 계속 한다.**

> 에러 때 뜨는 애들이 계속 떠있다. 이거 전부 가려두자. – css\ #signup

    function idConfirm() {

        const mustId = document.querySelector(".must\_id")

        const overlap = document.querySelector(".overlap")

        // 텍스트가 남아있어서 공백, 중복이 이중으로 뜨게 되는걸 방지

        mustId.style.display = "none"

        overlap.style.display = "none"

        // 1. userId 공백여부 확인

        // null, undefined, "" , 0은 false

        // 즉, userid가 빈 문자열일때 fuction을 실행한다는 의미

        if(!userid.value){ // 빈 문자열일 때!

            mustId.style.display = "block" /\* css설정 \*/

            return false //잘못된 값이므로! false 보냄. -> submit이 되지 않게 하는 억제기!

        } else { // 중복여부를 체크해야함

            if(!idCheck(userid.value)){  // id가 중복일 때

                overlap.style.display = "block" /\* css설정 \*/

                return false //잘못된 값이므로! false 보냄. -> submit이 되지 않게 하는 억제기!

            }

        }

        return true

// 비밀번호 정규식 체크하는 아이 (정규식은 따로 검색하는게 빠름)

    function pwdCheck(pwd){

     const reg = /^.\*(?=^.{8,15}$)(?=.\*\d)(?=.\*[a-zA-Z])(?=.\*[!@#$%^&+=]).\*$/;

       return reg.test(pwd)

    }

**///////////////// 자체 : register.js 빈칸에서 새로 시작하기**

 function pwd1Confirm() {

        const mustPwd1 = document.querySelector(".must\_pwd1")

        const regPwd = document.querySelector(".reg\_pwd")

        mustPwd1.style.display = "none"

        regPwd.style.display = "none"

        if(!pwd1.value){

            mustPwd1.style.display = "block"

            return false

        } else { //특문, 숫자, 문자 포함 8~15 여부 -> 자바스크립트 정규식 이용

                // js 정규표현식 검색해서 띄워봐야함

            if(!pwdCheck(pwd1.value)){ // 정규표현식에 맞지 않는다면

                regPwd.style.display = "block"

                return false

            }

        }

        return true

    }

    function pwd2Confirm() { //중복, pwd1이랑 같은지 여부

        const mustPwd2 = document.querySelector(".must\_pwd2")

        const same = document.querySelector(".same")

        mustPwd2.style.display = "none"

        same.style.display = "none"

        if(!pwd2.value){

            mustPwd2.style.display = "block"

            return false

        }else { //pwd1, 2 모두 값이 있는지 확인 / 그 다음에 같은지 확인

            if(pwd1.value && pwd2.value){

                if (pwd1.value !== pwd2.value){

                    same.style.display = "block"

                    return false

                }

            }

        }

        return true

    }

submitButton.addEventListener('click',function(){

    // 전송시키기

    const idConf = idConfirm()

    const pwd1Conf =pwd1Confirm()

    const pwd2Conf =pwd2Confirm()

    const fullnameConf =fullnameConfirm()

    const emailConf =emailConfirm()

    const telConf =telConfirm()

    if(idConf && pwd1Conf && pwd2Conf && fullnameConf && emailConf && telConf){ //모두 true 여야함

        document.signup.submit(); //signup이라 하는것을 submit시켜줌 - 보내준다

    }

// 다른버전

    let chkArray = [idConfirm(), pwd1Confirm(), pwd2Confirm(), fullnameConfirm(), emailConfirm(), telConfirm()]

    // for문을 이용하여 하나하나씩 보내준다

    let chkFlag = true

    for (const chk of chkArray){

        if(!chk){ //확인하는 아이들의 return 값이 false인경우

            chkFlag = false // 하나라도 false가 들어온 경우.

        }

    }

    if (chkFlag){ // 다 true가 뜨면 자료를 보냄.

        document.signup.submit()

    }

//////////////////////////////// 숙제 : select 이용해서 해봐라 (성별, 생년월일) ////////////////////////////////////

Register.js 안 모든 value 뒤에 replace 수정해라

**클래스**

자바에서 나오는 클래스와 동일함.

객체를 만들기 위한 설계도 / 틀

- 추상화? : 프로그램에 필요한 요소만 사용해서 객체를 표현하는 것

-> 결국에는 객체를 생성하는 것. 4번째 방법을 배운다…

        class Pet {

            // 생성자 함수 constructor

            constructor(nameVal, ageVal, catVal){

                this.name = nameVal

                this.age = ageVal

                this.cat = catVal

            }

아랫부분은 나머지와 동일.

객체를 실제 코드로 구현한 것 -> 인스턴스

        const pet = new Pet('구름',15,'페르시안')

        const pet2 = new Pet('별',10,'러시안')



//////////////// 클래스 완성!

- 상속

부모 클래스, 자식 클래스가 있을 때, 부모 클래스의 속성과 메소드를 그대로 자식 클래스도 사용할 수 있다.

Super : 부모클래스에 있는 생성자 함수를 호출한다

메소드는 별 다른거 없이 그대로 가져올 수 있음.

>> 인스턴스 선언을 해줘야 값 가져온 걸 사용할 수 있으니 주의!

- private

      class Square {

            #length //this.length를 private로 사용하겠다는 정의

                constructor(length){

                this.#length = length //this.length에 #을 붙임

            }

                getParameter(){

                    return 4\*this.#length

                }

                getArea(){

                    return this.#length \* this.#length

                }

            }

            const square = new Square(10)

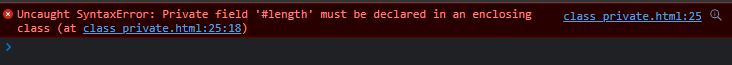
           square.length = -10 // 해버리면 length가 값이 변해버림. 즉 외부 방해없이 할 수 있는 private가 필요함.

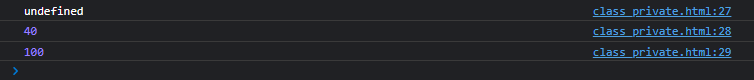
            //.#length를 해서 값을 바꾸려해도 바꿀 수 없다고 뜸.

           console.log(`${square.length}`); //private의 값이 되므로 undefined의 값이 떠버림.

            console.log(square.getParameter()); // 그래도 이미 값이 들어가 있어서 결과값은 잘 뜬다.

            console.log(square.getArea()); //





- 값을 지정하고 보내주는 get, set.

    <script> /\* get, set \*/

      class Square {

            #length

                constructor(length){

                this.#length = length

            }

                get length(){ //값을 보여줌

                    return this.#length

                }

                set length(length){ //값을 셋팅해줌

                    this.#length = length

                }

                get getParameter(){

                    return 4\*this.#length

                }

                get getArea(){

                    return this.#length \* this.#length

                }

            }

            const square = new Square(10) // -> get length를 보여주는 거임. this.length => this.#length

           console.log(`${square.length}`);

            //console.log(square.getParameter());

            console.log(square.getArea); // get, set는 함수가 아니고 단순 지정 및 넘겨주는 역할이기 때문에 옆에 괄호를 없애고 사용해줘야함.

- static

> 인스턴스 선언 없이 클래스에서 바로 실행이 가능해짐.

  class Pet {

static color = 'black' // static 속성 (=정적속성)

            //static 함수 (=정적 메소드)

            static namePrint(){

                console.log(`이름은 ${this.name} 입니다`);

            }

            static colorPrint(){

                console.log(`이름은 ${this.color} 입니다`);

            }

        }

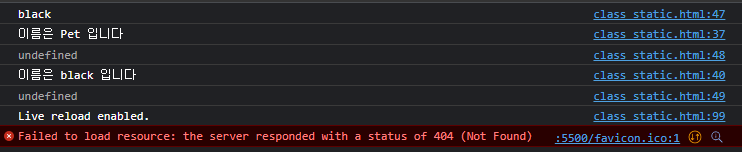
        console.log(Pet.color); //정적메소드 동작

        console.log(Pet.namePrint()); // 정적 함수 동작

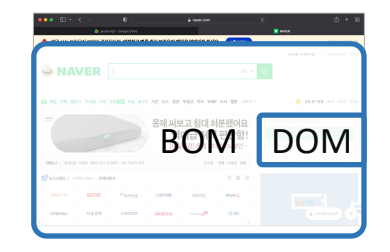
        console.log(Pet.colorPrint()); // 정적 함수 동작

        // Pet.namePrint에 나오는 값은 static으로 선언된 애들만 나올 수 있음.

            // static속성만이 값으로 활용되어 나올 수 있어짐.



왜 쓰냐? 클래스에 가깝게 포함되어서 관리되기가 쉽다 (따로 function을 안 만들어도 됨.)

브라우저 객체 모델 BOM

DOM과의 차이 : BOM 안에 DOM이 속해있음

- 브라우저 계층구조 : 최상위 객체 window, 그 하위에 있는 나머지들

Alert, confirm, prompt, setTimeout 는 윈도우 객체에서 돌아가는 애

        alert('h')// 윈도우라는 객체 안에 있는 애.

        window.alert('hee')// 이렇게 쓸 수 있으나, window는 생략가능

버거킹 다시 들어간다.

Js 라이브러리 사용 -> swipe 라는 애에 주로 사용 됨.

css에는 기본적으로 reset 코드가 들어간다 -> 완성된 css reset코드를 따오자

**오늘 공부한 내용**

**레지스터 폼**

**필수내용, input 조건에 빨간색 넣기**

**submitbutton 전송시키기**

**++homework : select 이용 성별, 생년월일 추가**

**register.js value 뒤에 replace 추가**

**아예 빈칸에서 register 만들어보기**

**클래스**

**class 형성, 상속 - super, private, get-set, static**

**BOM - dom**