**Javascript 공부 정리**

전채현

**배열**

const arr = [40,20,30,'40',11];

console.log(arr);

// 자바스크립트 객체를 하나하나 다 볼 수 있다 -> console.log을 장점

arr.sort();

// () -> 매개변수를 나타냄

// 자바스크립트의 arrow function

// func1은 요소와 인덱스를 콘솔로그에 출력해주라는 함수이다

const func1 = (element,index) => {

console.log(element,index)

};

// 이제 func1이 arr배열을 돌면서 요소와 인덱스를 콘솔로그에 출력하게 된다

arr.forEach(func1)

// document에서 selctorAll 하겠다 .image라는 selector를

// .image를 가져온다

const imageList = document.querySelectorAll('.image')

// 콘솔로그로 출력한다

console.log(imageList)

// foreach를 통해서 .image로 가져온 요소에 스타일 한번에 적용할 수 있다

imageList.forEach(img => { // 화살표 함수의 매개변수가 1개라면 () 생략 가능

img.style.border = '3px soild black'

});

// 문서의 특정요소를 불러와서 한꺼번에 작업을 진행할 수 있다

// 화살표 함수의 실행 코드가 한줄이라면 ()생략 가능

//imageList.forEach(img => img.style.borderRadius = '50%')

imageList.forEach(img => img.style.borderRadius = '70px')

// 이렇게 하면 이미지에는 인라인스타일로 적용이 된다

================================================

**css 선택자를 이용해서 요소를 불러오기**

document.getElementById('id') // 지정한 id를 가지느 요소 하나를 반환

document.querySelector('css선택자') // 지정한 선택자에 해당하는 요소 중 첫번째 요소 하나만

document.querySelectorAll('css선택자') // 지정한 선택자에 해당하는 모든 요소를 nodeList로 반환

1) 이름 나이 전화번호가 적혀있는 첫번째 줄의 칸 배경색을 #dadada로 변경하기

const headList = document.querySelector('thead')

for(let i = 0; i < headList.length; i++){

const th = headList[i]

th.style.backgroundColor = '#dadada'

} -> 기본 for문으로 돌려도 된다

console.log(headList)

css에서 하이픈을 사용하려면 여기서는 camelCase를 적용해서 해야한다

don`t forget camelCase

단일 객체이기 때문에 foreach문을 돌리지 않고 단독으로 style을 준 것이다

headList.style.backgroundColor = '#dadada'

2) 각 나이가 적혀있는 칸을 불러와서 내부 텍스트에 '살' 이라는 문자를 추가한다

const ageList = document.querySelectorAll('tbody>tr>td:nth-child(2)')

console.log(ageList)

ageList.forEach(td => {

td.style.color = 'blue'

td.style.textAlign = 'center'

td.innerText += '살' //ㄴ-> 기존의 것에 살을 더해주는 것이다

//td.append('살')

})

3) 전화번호가 적혀있는 칸에 마우스를 올리면 링크처럼 손가락모야응로 나타나도록 설정하세요

const cursor = document.querySelectorAll('tbody>tr>td:nth-child(3)')

console.log(cursor)

cursor.forEach(td =>{

td.style.cursor = 'pointer'

td.style.textDecoration = 'underline'

})

4) 이름 bold 처리

const nameList = document.querySelectorAll('tbody>tr>td:nth-child(1)')

console.log(nameList)

nameList.forEach(td =>{

td.style.fontWeight = 'bolder'

})

================================================

**요소불러오기**

<p>이름<input type="text" name="name" id="name"></p>

<p>나이<input type="number" name="age" id="age"></p>

<p>출생년도<input type="number" name="year" id="year"></p>

<p><button id="btn">입력</button></p>

<div class="result"></div>

<script>

// id를 이용해서 요소 불러오기

const name = document.getElementById('name')

// css선택자를 이용해서 요소 불러오기

const age = document.querySelector('#age')

const year = document.getElementById('year')

const btn = document.getElementById('btn')

const result = document.querySelector('.result')

console.log(btn) // HTML 태그만 출력

console.log({'btn' : btn}) // HTML 요소의 내부 상세 정보를 출력

const btnHandler = (event) => { // [버튼을 클릭하면] 작동하게 만들 함수를 작성

console.log(event)

console.log(event.target)

console.log(name.value) // element.property 요소.속성의 형식으로 값에 접근할 수 있다

console.log(age.value)

console.log(year.value)

console.log(name.type)

if(name.value == '' || age.value == '' || year.value==''){

alert('모든 값을 입력해주세요')

return

}

// 이 방식은 이클립스에서 사용이 불가능하다

const str = `${name.value}님은 ${year.value}생이고 ${age.value}살이고, ${age.value >= 20 ? '성인' : '미성년자'}입니다`

result.innerText = str

// == : 타입이 달라도, 값의 형태가 일치하면 일치로 판단

// === : 타입도 같고, 값도 같아야 일치로 판단

const str2 = name.value + '님은' +

age.value + '살이고, '+

(age.value >= 20 ? '성인' : '미성년자') + '입니다'

alert(str2)

}

btn.onclick = btnHandler // 버튼을 클릭하면 지정한 함수가 실행되도록

// 함수를 지금 호출하는 것이 아니므로, btnHandler()형식이 아니다

// 콘솔로그창에 PointerEvent가 발생한다

</script>

================================================

**정수를 입력받아서 자릿수를 거꾸로 배치하고 result태그 내 출력하기**

const num = document.getElementById('num')

const btnStr = document.getElementById('btnStr')

const btnNum = document.getElementById('btnNum')

const result = document.querySelector('.result')

const btnHandler = (event) =>{

// num.value의 값도 let으로 만들어서 사용해야 한다

let value = +num.value // Integer.parseInt(value)랑 동일함

let ans = 0 // 값을 변경해야 하므로 let

console.log(value)

while(true){

ans += (value % 10)

ans \*= 10

value /= 10 // /= 을 안해줌

value = Math.floor(value) // 소수점을 버리는 함수

if(value < 10){

ans += value

break

}

}

const str = ans + '입니다'

result.innerText = str

}

btnNum.onclick = btnHandler

btnStr.onclick = () => {

const value = num.value

var name = num.value

var reverseName = name.split('').reverse().join('')

// 입력받은 문자열을 글자하나씩 구분해서 뒤집고

// 다시 연결해서 하나의 문자열로 만듦

result.innerText = '' // innerText안에 한번 비워주기

for(let i = value.length-1; i > 0; i--){

console.log(value[i])

result.innerText += value[i]

}

// 뒤에서부터 한글자씩 result.innerText로 넣는다

console.log(reverseName)

}

===========================================

**필요한 요소 불러오기**

const children = document.querySelector('.form').children

console.log(children)

const button = document.querySelector('input[type="button"]')

const tbody = document.querySelector('tbody')

// 호출 함수 선언

const buttonClickhandler = () =>{

const product = document.querySelector('input[name="product"]')

const price = document.querySelector('input[name="price"]')

// category 자체가 value를 포함하기 때문에 문자열

const category = document.querySelector('select').value

const arr = price.value.split('')

console.log(arr)

let tr = document.createElement('tr')

let td1 = document.createElement('td')

td1.innerText = product.value

let td2 = document.createElement('td')

for(let i = 0; i < arr.length; i++){

if(i == 0) {

arr[i] += ','

}

td2.innerText += (arr[i])

}

td2.innerText += '원'

//td2.innerText = price.value.slice(0,1) + ',' + price.value.slice(1,4)+ '원'

// 앞에서 이미 value를 불어왔기 때문에 따라 value를 두번 붙일 필요는 없다

let td3 = document.createElement('td')

td3.innerText = category

// tr에 td넣기

tr.appendChild(td1)

tr.appendChild(td2)

tr.appendChild(td3)

// tbody에 tr 넣기

tbody.appendChild(tr)

// 값 콘솔로그로 확인하기

console.log(product.value)

console.log(price.value)

console.log(category)

// 값 지우기

product.value = ''

price.value = ''

document.querySelector('option').selected = 'selected'

// 입력이 끝나면 product로 돌아가라

product.focus()

}

// document.querySelector('input[type="button"]').onclick = () => {

// console.log(document.querySelector('select').value)

// document.querySelector('option').selected = 'selected'

// }

// 이벤트 함수 호출

button.onclick = buttonClickhandler

===========================================

**자바스크립트**

자바스크립트 -> Document Object -> Window Object

자바스크립트 탄생 -> 전송하는 데이터에 대한 유효성 검사의 필요성

inline 방식은 태그에 직접 자바스크립트를 기술하는 방식이다.

장점은 태그에 연관된 스크립트가 분명하게 드러난다는 점이다.

하지만 정보와 제어가 섞여 있기 때문에 정보로서의 가치가 떨어진다

(유지보수가 어렵기 때문이다)

<input type="button" onclick="alert('hello')" value="hello world">

onclick은 HTML문법이고 alert은 javascript문법인데 어떻게 작동할까??

-> onclick안에는 javascript가 들어간다고 약속을 했기 때문이다

script태그가 나오면 자바스크립트가 나온다는 것을 약속 / script는 일단 HTML태그이다

<script src="./script.js">

</script>

-> HTML의 정보로써의 가치가 더욱 높아진다

-> 유지보수 편리성 / 결합도 낮춤 / 브라우저 캐시를 줄여줄 수 있다

-> 이미 다운로드받은 파일을 가지고 있는데 js를 미리 다운받으면 서버쪽에 부담을 줄여줄 수 있다

-> 서버와 클라이언트가 주고 받는 데이터의 양을 줄여준다

-> script를 head 태그에 위치시킬 수도 있다. 하지만 이 경우는 오류가 발생한다.

-> 상단에 걸어두면 해당 태그를 찾을 수 없다 그래서 null을 발생시킴

해결방법

<script>

window.onload = function(){

var hw = document.getElementById('hw');

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

alert('hello world');

})

}

</script>

-> window.onload를 해주면 모든 문서를 다 다운로드한 후에 해당 function을 로드하겠다는 의미라서 작동이 된다

(이 경우는 head에 스크립트를 작성할 때만 하면 된다/ 스크립트를 바닥에 작성하면 따로 window.onload를 안해도 된다)

===========================================

**Object Model**

웹브라우저의 구성요소들은 하나하나가 객체화되어 있다. 자바스크립트로 이 객체를 제어해서 웹브라우저를 제어할 수 있게 된다.

이 객체들은 서로 계층적인 관계로 구조화되어 있다. BOM과 DOM은 이 구조를 구성하고 있는 가장 큰 틀의 분류라고 할 수 있다.

window -> (DOM, BOM, JavaScript)

각각의 태그들을 객체로 만들어두고 준비하고 있는 것이다

그래서 그 객체를 찾아서 제어할 수 있다 / 객체가 가리키는 태그를 제어할 수 있고 프로퍼티를 사용할 수 있고 메서드를 사용할 수 있다

객체를 통해서 브라우저를 제어할 수 있다

var imgs = document.getElementByTagName('img');

-> 이것을 객체화라고 한다

윈도우 : 전역객체, window frame을 제어하기 위한 객체

프로퍼티 - DOM(document) BOM(navigator, screen, location, frames,..) JavaScript(Object, Array..)

document 객체가 하는 역할은 웹 페이지의 문서를 제어하는 역할

Document Object Model Browser Object Model JavaScript Core (BW, GAS, Node.js..)

-> 자바스크립트를 통해 브라우저를 제어

// 자바스크립트 10대 에러

https://blog.meeta.io/10

BOM(Browser Object Model)이란 웹브라우저의 창이나 프래임을 추상화해서 프로그래밍적으로 제어할 수 있도록 제공하는 수단이다.

BOM은 전역객체인 Window의 프로퍼티와 메소드들을 통해서 제어할 수 있다.

전역객체 window

-> window 객체는 식별자 window를 통해서 얻을 수 있다. 또한 생략 가능하다

-> 변수가 함수에 소속되어 있지 않으면 전역변수가 된다 -> window객체의 프로퍼티를 만드는 것이다

사용자와 커뮤니케이션 하기

-> HTML은 form을 통해서 사용자와 커뮤니케이션할 수 있는 기능을 제공한다.

자바스크립트에는 사용자와 정보를 주고 받을 수 있는 간편한 수단을 제공한다

alert -> 경고찬이라고 부른다. 사용자에게 정보를 제공하거나 디버깅등의 용도로 많이 사용한다

-> 확인을 누르기 전까지 다음행동이 작동하지 않는다

confirm -> 확인을 누르면 true, 취소를 누르면 false를 리턴한다

<input type="button" value="localconfrim" onclick="func\_confirm();">

<script>

function func\_confirm(){

if(confirm('ok?')){

alert('ok');

}

else{

alert('cancel');

}

}

</script>

-> 조건문 처럼 사용할 수 있다

prompt -> 사용자가 입력한 값을 받아서 자바스크립트가 얻어낼 수 있는 것

function func\_prompt(){

if(prompt('id') === 'hi'){

alert('welcome');

}

else{

alert('fail');

}

}

===========================================

**Location 객체**

Location 객체는 문서의 주소와 관련된 객체로 Window 객체의 프로퍼티다.

이 객체를 이용해서 윈도우의 문서 URL을 변경할 수 있고, 문서의 위치와 관련해서 다양한 정보를 얻을 수 있다.

-> 보이지 않는 정보에 대해서도 객체화할 수 있다

현재 윈도우의 URL 알아내기

아래는 현재 윈도우의 문서가 위치하는 URL을 알아내는 방법이다.

console.log(location.toString(), location.href);

URL Pasring

location 객체는 URL을 의미에 따라서 별도의 프로퍼티로 제공하고 있다.

console.log(location.protocol, location.host, location.port, location.pathname, location.search -> ?뒤에 내용, location.hash -> #뒤에 내용)

http(protocol)://open.org(host):80(port)/module(pathname)/1?id=1(search)/#hash(hash)

===========================================

**URL 변경하기**

location.href = 'http://egoing.net';

-> 현재 문서를 해당 url로 이동한다

location = 'http://egiong.net'; -> 동일함

--> 다른 url로 이동시켜야 하는 경우에 사용한다

location.reload(); -> 현재 문서를 리로드하는 간편한 방법을 제공한다

Navigator 객체

브라우저의 정보를 제공하는 객체이다. 주로 호환성 문저등을 위해서 사용한다

console.dir(navigator);

크로스브라우징 -> 브라우저마다 다르게 작동하는 것을 말한다

자바스크립트 ->넷스케이프에서 만든 것이다

넷스케이프에서는 addEventListener를 사용하고 IE에서는 attachEvent를 사용해야 했다

-> 그래서 웹 표준이 나왔다 -> 같은 기능이면 같은 이름을 사용하도록 함

-> 그래서 크로스브라우징이 나옴 -> navigator객체를 사용해서 확인한다

-> 해당 브라우저의 특성을 확인하려면 console.dir(navigator);를 통해서 확인한다

console.dir(navigator.appName); -> netscape라고 뜬다

console.dir(navigator.appVersion)

-> 브라우저의 버전을 의미한다.

console.dir(navigator.userAgent)

-> 브라우저가 서버측으로 전송하는 user-agent-http 해더의 내용이다.

console.dir(navigator.platform)

-> 운영체제 정보

===========================================

**기능 테스트**

navigator 객체는 브라우저 호환성을 위해서 주로 사용하지만 모든 브라우저에 대응하는 것은

쉬운 일이 아니므로 기능 테스트를 사용하는 것이 더 선호되는 방법이다

Ex) Object.keys라는 메소드를 객체의 key값을 배열로 리턴하는 Object의 메소드이다.

이 메소드는 ECMAScript5에 추가되었기 때문에 오래된 자바스크립트외는 호환되지 않는다.

창 제어

window.open 메소드는 새 창을 생성한다. 대부분의 브라우저는 탭을 지원하기 때문은 window.open은 새 창을 만든다

================================================

// slider.style.marginLeft = '-400px' // margin-left , css에서 -가 들어가는 속성은 camelCase로 변환

================================================

let value = 0 // 값이 변해야 하는 값이면 let

const unit = 1920 // 변하지 않는 값은 const

================================================

<script>

// 한줄 주석

/\*

범위 주석

\*/

// 변수 선언

n1 = 10;

var n2 = 20; // n2는 객체에 대한 이름이다 데이터를 박스로 감싸서 참조하는 이름이 n2이다

// 자바스크립트는 wrapper클래스만 가지고 있다

// wrapper클래스와 wrapping 방식

// var x = 3 -> var x = new Number(3)으로 되는 것이다

let n3 = 30; // 값이 변하는 변수면 let 레퍼런스타입

const n4 = 40; // 값이 변하지 않는 변수 const 프리미티브타입

const word = 'hello'; // 대입되는 값의 자료형에 따라서 자동으로 설정된다

// 함수선언

function func1(){

console.log('함수1')

}

const func2 = function(){ // 자바스크립트의 함수는 그 자체로 객체 취급 된다

console.log('함수2') // 자바처럼 함수형 인터페이스 객체가 아니라

} // 함수 자체로 매개변수로 전달할 수 있다

console.log(n1,n2,n3,n4);

// 함수 호출

func1();

func2();

const arr = [10,20,30,40,50]; // {}가 아니고 []로 배열을 표현한다

// 배열이 리스트처럼, 컬랙션처럼 함수를 가진다

arr.forEach(function(num){

console.log(num)

// arr안에 있는 각각의 요소를 num이라고 할때,

//그 num의 값을 콘솔 창에 출력 (num에 var는 사용불가)

});

setInterval(func2, 1000);

// 만들어진 함수를 전달해서 사용한다

// 시간 간격을 주고 출력한다

// 함수를 객체취급해서 전달할 수 있다

console.log(typeof n1)

console.log(typeof word)

console.log(typeof func2)

console.log(typeof arr)

</script>

===========================================

**innerText, innerHTML**

element.innerText;

이 속성은 element 안의 text 값들만을 가져옵니다.

element.innerHTML;

innerText와는 달리 innerHTML은 element 안의 HTML이나 XML을 가져옵니다

element.innerText = "<div style='color:red'>A</div>";

element.innerText에 html을 포함한 문자열을 입력하면, html코드가 문자열 그대로 element안에 포함됩니다.

element.innerHTML = "<div style='color:red'>A</div>";

위와 같이 element.innerHTML 속성에 html코드를 입력하면, html element가 element안에 포함되게 됩니다.

위 예제에서 'innerHTML()'을 클릭하면, 입력된 html태그가 해석되어 빨간색A 가 나타나는 것을 확인 할 수 있습니다.

===========================================

**append, appendChild**

append와 appendChild는 모두 부모 노드에 자식 노드를 추가하는 메서드

append

-> 노드객체나 DOMString('text')을 사용할 수 있다.

-> 또한 한번에 여러 개의 자식 요소를 설정할 수 있다

-> return 값을 반환하지 않는다

노드 객체 사용 예시

const p = document.createElement('div')

const c = document.createElement('p')

p.append(c)

//결과

<div><p></p></div>

문자열 (DOMString) 사용 예시

const p = document.createElement('div')

p.append('append 예시')

//결과

<div>append 예시</div>

appendChild

-> 오직 Node 객체만 받을 수 있다

-> 한번에 오직 하나의 노드만 추가할 수 있다

-> return 값을 반환합니다

문자열(DOMString) 사용 예시

const p = document.createElement('div')

p.appendChild('텍스트')

-> 이러면 에러 발생

return 값 반환

const p = document.createElement('div')

const s = document.createElement('span')

console.log(document.body.appendChild(div))

===========================================

<script>

const slider = document.querySelector('.slider')

// let 값 변경된는 변수

let value = 0

// const는 고정값

const unit = 1920

function move() {

value -= unit

if(value <= -7680) {

value = 0

}

slider.style.marginLeft = value + 'px'

}

setInterval(move, 3000);

const menuList = document.querySelectorAll('.list')

const contentList = document.querySelectorAll('.content\_img')

for(let i = 0; i < menuList.length; i++){

const list = menuList[i]

// list onclick하면 해당 함수를 실행

list.onclick = function(){

// 모든 요소에 hidden 추가

contentList.forEach(element => element.classList.add('hidden'))

// i번째에 hidden 제거

contentList[i].classList.remove('hidden')

// 모든 요소에 selectedList제거

menuList.forEach(element => element.classList.remove('selectedList'))

// i번째에 selectedList추가

list.classList.add('selectedList')

}

}

</script>

===========================================

**배열 객체**

push/pop 메서드를 이용한 데이터 관리 : stack

<script>

var nums = new Array();

nums.push(5);// nums[0] = 5;

nums.push(10);// nums[1] = 10;

nums.push(21);// nums[2] = 21;

// pop은 맨 위에 적재된걸 호출하면서 삭제

var n1 = nums.pop();

var n2 = nums.pop();

var n3 = nums.pop();

alert(nums)

</script>

배열 객체 초기화

var nums = new Array();

var nums = new Array(5);// 5개를 만듬

var nums = new Array(5,10,21);// 방이 3개짜리를 만듬

var nums = new Array(5,10,21,"hello") // 이렇게도 사용가능함

그래서 console.log(typeof nums[3]); 배열 index의 타입을 확인해야한다

var nums = new Array(5,10,21,"hello", new Array(2,3,4));

-> js에서 배열은 모든 값을 담을 수 있기 때문에 주의해야 한다

===========================================

**정적인 객체 정의와 동적인 객체 정의**

c++, c#, Java -> class (클래스를 만들고 생성자를 통해서 객체를 생성)

javascript -> 객체를 만들고 객체의 타입을 정의 (prototype, class)

Object 객체의 Expand Object

키를 기반으로 데이터를 저장해야 할 때

var exam = new Object();

exam.kor = 30;

exam.eng = 50;

exam.mat = 40;

--> 기본적으로 이방법을 사용하는 것이 바람직

키를 이용한 데이터 관리 : Map

var exam = new Object();

exam["kor"] = 30;

exam["eng"] = 40;

exam["mat"] = 50;

===========================================

**Javascript 데이터 객체와 JSON 생성 방법**

JavaScript Object JavaScript Object Notation(JSON)

Booleanvar n = new Boolean(true);var n = true;

Numbervar n = new Number(3);var n = 3;

Stringvar n = new String("hello");var n = "hello";

Arrayvar n = new Array();var n = [];

Object var n = new Object();var n = {};

var exam = {"kor" : 30, "eng" : 41, "mat":50}

var ar = [3,4,5,6,7,[4,1,3]]

console.log(exam.kor + exam.mat)

console.log(ar[5][1])

===========================================

**JSON의 중첩 표현**

중첩된 데이터의 구조도 쉽게 표현할 수 있다

var notices = [

{"id" : 1, "title" : "hello~"},

{"id" : 2, "title" : "hello11~"},

{"id" : 3, "title" : "heww~"}

];

---> 이렇게 중첩적으로 사용할 수 있다

===========================================

<script>

// 이럴 경우 한칸 띄우고 역슬러쉬해서 배열로 인식하게 만듬

var data = '[ \

{"ty\_total":null,"ty\_care":25868,"sy\_kj":null,"sy\_total":null,"ty\_kh":null,"death":2773,"s\_death":769,"jgk\_cont\_gh":null,"ty\_kj":null,"jgk\_overseas\_gh":null,"sy\_kh":null,"recover":324448,"sn\_recover":488,"t\_hj":353089,"n\_hj":1190,"s\_hj":115417,"s\_recover":103912}, \

{"ty\_total":null,"ty\_care":25740,"sy\_kj":null,"sy\_total":null,"ty\_kh":null,"death":2766,"s\_death":764,"jgk\_cont\_gh":null,"ty\_kj":null,"jgk\_overseas\_gh":null,"sy\_kh":null,"recover":323393,"sn\_recover":258,"t\_hj":351899,"n\_hj":1423,"s\_hj":114962,"s\_recover":103424}, \

{"ty\_total":null,"ty\_care":25195,"sy\_kj":null,"sy\_total":null,"ty\_kh":null,"death":2745,"s\_death":756,"jgk\_cont\_gh":null,"ty\_kj":null,"jgk\_overseas\_gh":null,"sy\_kh":null,"recover":322536,"sn\_recover":804,"t\_hj":350476,"n\_hj":1508,"s\_hj":114442,"s\_recover":103166} \

]';

eval('var x = 40;');

console.log(x);

// 데이터를 파싱하기

eval("var ar =" + data + ";");

console.log(ar[0].ty\_total)

</script>

===========================================

<script>

var data = {"id":1, title:"aaa"};

console.log(data.title)

console.log(data["id"])

// 변수도 문자열 형태도 되어있지 않으면 파싱을 할 수 없다

var data2 = JSON.parse ('{"id":3, "title":"bbbb"}');

console.log(data.title)

// data3같은 데이터를 위에 처럼 만들어 줄 때

var data3 = {id:4, title:"ccc"};

var json = JSON.stringify(data3);

</script>

===========================================

var x = 3;

var y = 3;

document.write(x == y) -> 값이 같은지 확인

document.write(x === y) -> 주소가 같은지 확인

-> true, true 나옴

===========================================

<script>

var ar = ["철수", "영희", "맹구","동천"]

var ob = {id:1, title:"hello",writeid:"newlec"}

for (var i in ar){

document.writeln(ar[i])

}

// i : key 나옴

// ob[i] : value가 나옴

for(var i in ob){

document.writeln(i)

}

</script>

===========================================

**함수를 객체로 받아서 사용가능**

var add = new Function("x,y", "return x+y;");

document.writeln(add(3,4))

-> var add = function (x,y){

return x+y;

};

-> function add (x,y){

return x+y;

}

=========================================

<script>

// 매개변수 의미 없다 값을 받는 그릇의 의미 X

// 참조하는 x, y일뿐이다

var add = function(x,y){

// 가변적인 매개변수

// 전달된 매개변수의 갯수를 파악할 수 있다

alert(arguments.length);

alert(typeof arguments[4])

return x+y;

}

document.writeln(add(16,3,6,3,"hello"))

</script>

===========================================

**변수의 가시영역과 전역변수**

<script>

// 자료형을 선언하지 안흐면 전역변수가 된다

a = 1 // window.a = 1

// 지역변수가 전역변수보다 우선순위가 높기 때문에 같은 변수라도

// 지역변수로 지정된 값이 출력된다

var a = 3

{ // 스크립트에서는 중괄호가 변수 영역을 나누지 않음

var a = 4

}

// 함수안에서 사용된 지역변수는 함수안에서만 사용가능

// 스크립트에서는 함수영역만 변수 영역을 나눔

alert(a)

</script>

===========================================

**변수 생명주기와 클로저(Closure)**

<script>

function f1(){

var a = 1

// f2가 클로저이다

// 자원을 묶어버린다

return function f2(){

return a;

}

}

var f = f1();

var a = f();

alert(a)

</script>

===========================================

**최상위 객체인 window 객체**

window 객체의 members : 사용자와 상호작용을 위한 기본 도구

대화상자를 이용한 출력 : alert()

대화상자를 이용한 입력 : prompt()

<script>

var x = prompt("x값 입력 ",0)

var y = prompt("y값 입력 ",0)

x = parseInt(x)

y = parseInt(y)

// return parseInt("abc") -> returns NaN

// return parseInt("12abc") -> returns 12

// 문자열로 출력

alert(x+y)

var answer;

answer = confirm("정말 삭제하시겠습니까")

if(answer){

alert("삭제되었습니다")

}

</script>

===========================================

**스크립트는 페이지가 읽혀질 때 실행**

**onclick은 이벤트가 발생할 때 실행**

<input type="button" name="" id="" value="클릭" onclick="printResult();">

<span onclick="alert('hello');">스팬입니다</span>

<script>

function printResult(){

var x = prompt("x값 입력 ",0)

var y = prompt("y값 입력 ",0)

x = parseInt(x)

y = parseInt(y)

// return parseInt("abc") -> returns NaN

// return parseInt("12abc") -> returns 12

// 문자열로 출력

alert(x+y)

var answer;

answer = confirm("정말 삭제하시겠습니까")

if(answer){

alert("삭제되었습니다")

}

}

</script>

===========================================

HTML 파일 - load -> Document Object (객체 트리 in memory) - show -> 사용자 인터페이스

input 타입은 속성에 value가 있어서 이렇게 사용할 수 있는데,

span의 경우에는 속성에 value가 없어서 이렇게 사용 불가능

btnPrint.value = x+y;

그래서 spand같은 경우에는 innerText를 사용

span1.innerText = x+y;

====================================

**btnPrint.onclick = printResult;**

-> 뒤에 괄호를 붙이면 함수를 대입하는 것이 아니라 호출하는 것

<input type="button" name="" id="" value="클릭" id="btnPrint">

<span id="span1" onclick="alert('hello');">스팬입니다</span>

<script>

function printResult(){

var x = prompt("x값 입력 ",0)

var y = prompt("y값 입력 ",0)

x = parseInt(x)

y = parseInt(y)

// return parseInt("abc") -> returns NaN

// return parseInt("12abc") -> returns 12

// 문자열로 출력

//alert(x+y)

// input 타입은 속성에 value가 있어서 이렇게 사용할 수 있는데

// span의 경우에는 속성에 value가 없어서 이렇게 사용 불가능

btnPrint.value = x+y;

// 그래서 spand같은 경우에는 innerText를 사용

//span1.innerText = x+y;

var answer;

answer = confirm("정말 삭제하시겠습니까")

if(answer){

alert("삭제되었습니다")

}

}

btnPrint.onclick = printResult;

</script>

====================================

위에 스크립트를 헤더쪽에 놔두면 오류가 발생하는 이유

- btnPrint가 선언되어 있는데 읽어오면서 값은 없으니까

- 버튼 객체가 만들어지기 전에 사용하려고 해서 오류가 발생

====================================

<input type="button" value="출력" id="btn-print">

- html에서는 명명규칙이 btn-print로 해야한다

- 그러면 자바스크립트에서 불러올 때 문제가 발생하기 때문에

var btnPrint = document.getElementById('btn-print')로 해서 변환해서 가지고 와야 한다

<input type="button" name="" id="" value="클릭" id="btn-print">

<span id="span1" onclick="alert('hello');">스팬입니다</span>

<script>

function printResult(){

// document가 보따리 역할

var btnPrint = document.getElementById('btn-print')

var x = prompt("x값 입력 ",0)

var y = prompt("y값 입력 ",0)

x = parseInt(x)

y = parseInt(y)

btnPrint.value = x+y;

}

function init(){

btnPrint.onclick = printResult;

}

window.onload = init;

</script>

====================================

여기서 변수를 전역변수로 빼고 싶을건데 바람직한 방법이 아니다

====================================

**스크립트를 사용하는 기본 구조**

<script>

function printResult()// 명명할 이유가 없는 함수

{

var btnPrint = document.getElementById('btn-print')

}

function init()// 명명할 이유가 없는 함수

{

btnPrint.onclick = printResult;

}

window.onload = init;

</script>

====================================

**익명 함수를 이용한 이벤트 핸들링**

<script src="./js/2.js">

로 js코드를 분리해서 사용가능

</script>

<script src="./js/index.js"></script>

이런식으로 스크립트를 여러개 열어버리면 앞에 function이 덮어씌워지는 경우가 발생한다

여러 스크립트 파일을 함께 사용할 때 초기화 함수의 문제

window.onload = function(){ alert('test1') }

window.onload = function(){ alert('test2') }

// addEventListener로 누적해서 사용가능

// 이벤트를 바인딩할 때 addEventListener를 사용가능

->> window.addEventListener("load", function(){

alert('test1')

});

====================================

**문서를 구성하는 노드들의 계보를 표현하는 방법 : 부모 / 자식**

childNodes를 이용한 노드 선택

<div class="box">

<input>

<input>

</div>

const box = document.querySelector('.box')

const input = box.childNodes[0]

-> childNodes는 주석도 자식으로 인식한다. 빈 공백도 하나의 노드로 인식해서 붙여사용하지 않으면 원하는 위치에 넣기 힘들다

그래서

children을 사용한다

-> children은 태그형태로 되어있는 것을 자식으로 인식한다. 그래서 공백과 무관하게 태그만 인식해서 자식으로 가져온다

====================================

**문서를 구성하는 Node의 종류**

DocumentType -> <!DOCTYPE>

Element -><p>

Attr->rows="", cols=""

Entity->lt, gt, nbsp

EntityReference->& ;

Text->태그안에 들어갈 문자

Comment-><!-- -->

CDATASection-><![CDATA[<,>,]]>

Notation->속성의 값 "#ffff00", "10px"

====================================

HTML 태그의 속성과 객체의 속성은 거의 일치

<input class="src-input" list="img-list">

<datalist id="img-list">

<option value="img1.jpg">img1</option>

<option value="img2.jpg">img2</option>

<option value="img3.jpg">img3</option>

</datalist>

-> 이렇게 하면 input타입에서도 select처럼 사용할 수 있다

====================================

txt1.style.width = '100px' -> 이렇게 줘야한다 뒤에 px빼먹으면 안됨

txt1.style.["border-color"] = img.value

txt1.stylw.borderColor = img.value 이렇게 두 가지 방식으로 지정할 수 있다

console.log(img.className) -> class는 예약어라서 바로 사용이 안되서 className으로 줘야지 사용가능하다

====================================

div.appendChild(txtNode)

div.removeChild(txtNode) -> 이런식으로 append를 할 수 있고, remove를 할 수 있다

====================================

menuList.innerHTML += '<li><a href="">' + key + '</a></li>'

menuList를 포함하는 안쪽에 태그라는 문자열을 넣게 되면 그걸 읽어서 해당하는 노드객체를 만들어준다

간단한 삭제기능

delBtn.onclick = function() {

let li = menuList.children[0]

menuList.removeChild(li)

};

-> menuList.remove() -> 이러면 자기 자신을 지우라는 뜻이 된다

-> 따로 클래스를 지정해주지 않는다면 remove()함수를 사용하면 자기 자신을 지우게 된다

append : 문자열을 그대로 삽입할 수 있다. 그리고 여러개의 노드를 입력 받을 수 있다

appendChild : 객체를 삽입, 단일로 입력

====================================

**노드 복제 및 템플릿 복제**

qureySelector는 첫번째 객체를 먼저 가지고 온다

const tr = document.querySelector('tbody tr')

const clone = tr.cloneNode( flase : 껍데기만 가져오기, true : 내용까지 가져오기 )

tbody.appendChild(clone) -> 이렇게 삽입해준다 복사해서

const tds = clone.querySelectorAll('td')

tds[0].innerHTML(textContent) -> 이렇게 두개 사용가능

tds[0].textContent = notices[0].id;

tds[1].textContent = notices[1].title;

tds[2].textContent = notices[2].regDate;

tds[3].textContent = notices[3].writerId;

tbody.appendChild(clone)

**템플릿**

<template>

<tr>

<td></td>

<td></td>

<td></td>

</tr>

</template>

-> 이렇게 묶어두면 화면에는 보이지 않는다

const template = document.querySelector('template')

const cloneNode = document.importNode(template.content, true) -> template안에 내용을 전부 다 가지고 오겠다

const tds = clone.querySelectorAll('td')

tds[0].innerHTML(textContent) -> 이렇게 두개 사용가능

tds[0].textContent = notices[0].id;

tds[1].textContent = notices[1].title;

tds[2].textContent = notices[2].regDate;

tds[3].textContent = notices[3].writerId;

tbody.appendChild(clone)

====================================

-> **스크롤 로딩**

<%@ page language="java" contentType="text/html; charset=UTF-8"

pageEncoding="UTF-8"%>

<%@ include file="header.jsp" %>

<div class="root"></div>

<script>

// 페이지를 새로고침하기 전까지는 유지되는 변수

let scrollCount = 0

const root = document.querySelector('.root')

const body = document.body

function getBoardData(){

const cnt = scrollCount \* 5

const url = cpath + '/getBoardData/' + cnt

const opt = {

method : 'get'

}

const pro = fetch(url, opt)

.then(resp => resp.json())

.then(json => {

console.log(json)

const dom = getDom(json)

console.log(root)

// body.onscroll = function(event){

// const ob = {

// clientHeight : event.target.clientHeight,

// scrollHeight : event.target.scrollHeight,

// scrollTop : event.target.scrolly,

// clientTop : event.target.clientTop

// }

// console.log(ob)

// }

window.onscroll = function(){

const ob2 = {

//x : window.pageYOffset,

scrollTop: document.scrollingElement.scrollTop,

clientHeight : document.scrollingElement.clientHeight,

scrollHeight: document.scrollingElement.scrollHeight

}

console.log(ob2)

const flag = ob2.scrollTop + ob2.clientHeight === ob2.scrollHeight

const num = scrollCount

let dom2 = ''

if(flag){

console.log('추가 불러오기')

console.log(scrollCount)

console.log(num)

// scrollCount += 5

// for(let i = num ; i< num + 5; i++){

scrollCount += 5

json.forEach(dto =>{

dom2 += '<div class="item">'

for(key in dto){

dom2 += '<div>' + dto[key] + '</div>'

}

dom2 += '</div>'

root.innerHTML += dom2

})

// }

}

}

root.innerHTML += dom

})

return pro

}

function getDom(json){

let dom = ''

// dom += '<div class="root">'

json.forEach(dto =>{

dom += '<div class="item">'

for(key in dto){

dom += '<div>' + dto[key] + '</div>'

}

dom += '</div>'

})

// dom += '</div>'

return dom

}

</script>

<script>

window.onload = function(){

const boardData = getBoardData()

}

</script>

</body>

</html>

====================================

**노드 삽입과 노드 순회**

tbody의 첫번째 노드를 가져오는 방법

const tbody = document.querySelector('tbody')

cosnt current = tbody.firstElementChild -> 이렇게 가져올 수 있다

current로 선택한 것의 다음 노드를 선택하는 방법

downBtn.onclick = function() {

const nextNode = current.nextElementSibling

if(nextNode == null){

alert('다음 노드는 없습니다')

return

}

tbody.removeChild(nextNode)

tbody.insertBefore(nextNode, currentNode)

// tbody.insertAdjacentElement('beforebegin', nextNode)

}

upBtn.onclick = function() {

const preNode = current.previousElementSibling

if(nextNode == null){

alert('다음 노드는 없습니다')

return

}

tbody.removeChild(currentNode)

tbody.insertBefore(currentNode, preNode)

}

target.insertAdjacentElement(position, element)

<!-- beforebegin -->

<p>

<!-- afterbegin -->

foo

<!-- beforeend -->

</p>

<!-- afterend -->

====================================

**일괄 삭제 , 노드 자리 바꾸기**

<input type="button" class="del-btn">

<input type="button" class="swap-btn">

<input class="overall" type="checkbox">

const allCheckbox = document.querySelector('.overall')

-> checkbox의 value는 기본값이 on이다

-> 전체 선택

swapBtn.onchange = function(){

const inputs = tbody.querySelectorAll('input[type="checkbox"]')

for(let i = 0; i < inputs.length; i++){

inputs[i].checked = allCheckbox.checked

}

}

-> 일괄 삭제

delBtn.onclick = function(){

const inputs = tbody.querySelectorAll('input[type="checkbox"]:checked')

for(let i = 0; i < inputs.length; i++){

inputs[i].parentElement.parentElement.remove()

}

}

swapBtn.onclick = function(){

const inputs = tbody.querySelectorAll('input[type="checkbox"]:checked')

if(inputs.length != 2) {

alert('2개만 선택해야 합니다')

return

}

const trs = [];

for(let i = 0; i < inputs.length; i++){

trs.push(inputs[i].parentElement.parentElement)

}

const cloneNode = trs[0].cloneNode(true) // 안에 내용을 전부 복사

trs[1].replaceWith(cloneNode)// 복사한 것을 1번째 넣겠다

trs[0].replaceWith(trs[1])// 빈자리에 채워 넣겠다

}

====================================

**정렬**

let titleSorted = false;

titleTd.onclick = function(){

items.innerHTML = ''

if(!titleSorted)

items.sort(function (a, b) {

titleSorted = true

if( a.value > b.value)return 1;

else if( a.value < b.value) return -1;

else if( a.value == b.value) return 0;

}

else

items.reverse();

화면 출력해주는 함수() -> 동작하게 해서 정렬될때마다 출력되야 하니까

}

====================================

**이벤트 객체**

-> 이벤트에서 발생한 부가정보를 제공

클릭했어요 -> 어느 위치를 ? -> 누굴 클릭함?

키가 눌렸어요 -> 어디에서? -> 어떤 키가 ? -> 어떤 키와 함께?

====================================

**이벤트가 발생한 객체는 event.target**

클릭한 이미지의 위치 정보 출력하기

const imgList = document.querySelectorAll('.img')

imgList.forEach(img =>{

img.onclick = function(event){

currentImg.src = event.target.src

}

})

삭제하기

const delBtnList = document.querySelectorAll('.delBtn')

delBtnList.forEach(del => {

del.onclick = function(event){

const idx = event.target.dataset.idx

location.href= cpath + '/delete/' + idx

}

})

====================================

**이벤트 버블링 => for문을 돌리면서 낭비되는 메모리를 줄여줄 수 있다**

const imgList = document.querySelector('.img-list')

imgList.onclick = function(event){

if(event.target.nodeName != 'IMG')

return

else

currentImg.src = event.target.src

console.log('test')

}

====================================

**같은 부모를 공유하지만 다른 행위를 해야 하는 경우**

addButton.onclick = function(e) {

e.stopPropagation()// 부모로 버블링 되는 것을 막기위한 것

const img = document.createElement('img')

img.src = ''

currentImg.insertAdjacentElement('afterend', img);

}

====================================

**특정 태그를 찾는 방법**

const tbody = document.querySelector('tbody')

tbody.onclick = function(event){

let target = event.target

if(target.nodeName != "INPUT")

return

if(target.classList.contains('sel-button'))

// tr을 찾는 방법

let tr = target.parentElement;

for( tr.nodeName != 'TR'; tr = tr.parentElement;)

else if(target.classList.contains('edit-button'))

else if(target.classList.contains('del-button'))

}

====================================

**엘리먼트 노드의 기본행위 막기**

event.preventDefault() -> 이벤트로 타고 오는 것에 대한 기본 기능을 막겠다

click 이벤트 트리거 구현 코드

파일 선택 상자를 대신하는 <span> 태그 사용

<div>

<input type="file" class="gallery-file">

<span class="file-button">파일선택</span>

const fileBtn = document.querySelector('.file-button')

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

let event = new MouseEvent('click', {

'view' : window,

'bubbles' : true,

'cancelable' : true

})

const file = document.querySelector('.gallery-file')

file.dispatchEvent(event)

})

====================================

**클릭한 위치로 박스 옮기기**

const container = document.querySelector('.container')

const box = document.querySelector('.box')

container.onclick = function(event){

console.log(event.x, event.y)

box.style.position = 'absolute'

box.style.left = event.x + 'px'

box.style.top = event.y + 'px'

}

자바스크립트 만들기

1. 이벤트를 등록할 요소를 문서에서 불러온다

- document.querySeletor

2. 특정 조건에 의해 호출될 함수를 미리 정의한다

- const handler = function(event)

3. 요소에게 함수를 연결한다

- element.onclick

=====================================