1 HTML의 form과 action의 개념

action 속성)

폼 데이터를 서버로 보낼 때, 데이터가 도착할 URL 명시

<form action=“URL”> 로 사용한다.

method )

<form> 태그의 method 속성은 폼 데이터(form data)가 서버로 제출될 때 사용되는 HTTP 메소드

GET,POST 두가지의 개념을 사용

GET )

URL에 폼 데이터를 추가하여 서버로 전달하는 방식

GET 방식의 HTTP 요청은 브라우저에 의해 캐시되어 저장된다. 또, 쿼리 문자열에 포함되어 전송되기 때문에 길이의 제한이 있다.

보안상 취약점이 존재하기 때문에, 중요한 데이터는 POST 방식을 사용하여 요청하는것이 좋다.

POST)

폼 데이터를 별도로 첨부하여 서버로 전달하는 방식

브라우저에 의해 캐시되지 않기 때문에, 브라우저 히스토리에도 남지않는다.

쿼리문자열과는 별도로 전송된다. -> 길이의 제한도 없고, 보안성이 높음

사용 방법)

<form method=“get 또는 post”>

parameter (name, value)

name = “value” 형식.

name : 속성의 이름 (href, src 등)

value : 속성의 값 (url 등)

예)

<a href=“www.naver.com”>

2 Javascript

* DOM Manipulation

Document Object Model. 문서를 조작할 수 있도록 객체 형태로 표현한 것

* HTML 요소의 속성이나 내용을 변경
* 새로운 요소를 추가 또는 기존 요소를 삭제 등

요소 선택 방법)

* getelementByID

html

<div id=“Earth”>Earth </div>

JavaScript

const Earth = document.getElementById (“Earth”);

Earth.textContent= “Mars” ;

* getelementsByClassName

html

<div class=“planet”>Earth </div>

<div class=“planet”>Mars </div>

JavaScript

const planetElements = document.getElementByClass (“planet”);

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

planetElements[i].textContent = “Hello, DOM!”;

}

Onclick 과 addEventListner 의 차이

Onclick)

하나의 이벤트만 적용 가능. 여러개를 적용하면 덮어써진다.

addEventListner)

여러개의 이벤트에 적용 가능

사용 예)

<button id = “button1”>button</button>

javascript

let buttonClick = document.querySelector(“#button1”)

buttonClick.onclick = function( ) {

console.log (‘button clicked”)

}

* Event loop

CallStack , CallBack Queue 를 감시하는 역할.

CallBack Queue 에 Task 가 존재하고, CallStack 이 비어있다면 Task 를 넣어주는 일

* queryselector vs queryselectorAll

querySelector)  
지정된 선택자와 일치하는 Document 의 첫 번째 element 를 반환.

일치하는 요소가 없으면 null 반환

사용 예)

const event = document.querySelector(‘#button1’);

querySelectorAll)

일치하는 element list 를 나타낸다.

- Node의 속성

노드 : 정보가 저장되는 단위.

노드의 종류)

문서 노드 : HTML 문서 전체를 나타냄

요소 노드(element node) : 모든 HTML 요소는 요소 노드이며, 속성 노드를 가질 수 있는 유일한 노드

속성 노드(attribute node) : 모든 HTML 요소의 속성은 속성 노드이며, 요소 노드에 관한 정보를 가지고 있음

텍스트 노드(text node) : HTML 문서의 모든 텍스트는 텍스트 노드

주석 노드(comment node) : HTML 문서의 모든 주석은 주석 노드임.

노드 간의 관계)

루트노드 <html>

첫번째 자식 노드 : 요소 노드 <head>

마지막 자식 노드 : 요소 노드 <body>

<head> , <body> 는 형제 노드

Javascript 문법

var와 let의 차이

var : 함수 단위 스코프

* 변수를 블록을 고려하지 않고, 어디에서나 접근할 수 있는 변수를 선언한다.
* 중복, 재할당 가능
* 함수 내부에서 선언되면 지역 변수, 외부에서 선언되면 전역 변수

var X = 5;

console.log(X) ; -> 5

let : 블록 스코프의 범위를 가지는 지역 변수를 선언한다.

* 중복 불가능
* 재할당 가능

let a = 10;

console.log(a) ; -> 10

const (상수를 선언)

* 중복 불가능
* 재할당 불가능

const b = 15;

console.log(b); -> 15

호이스팅)

선언 위치와 상관 없이 최상단으로 옮겨진것처럼 처리되는 것

변수 초기화는 위로 끌어올려지지 않음

ES5의 Array 관련 메소드 (최소 4개 이상)

1. array.forEach : 배열 전체를 돌며 배열의 요소에 어떤 작업을 수행하고싶을 때

예)

const numbers = [1,2,3,4];

numbers.forEach (function ( number, index) {

console.log( ‘Index ${index} : ${number}’ ) ;

});

돌면서

Index 0 : 1

Index 1: 2

Index 2: 3

Index 3: 4

를 출력한다.

const fruits = [‘apple’ , ‘banana’, ‘cherry’];

fruits.forEach(function(fruit) {

console.log(fruit.toUpperCase());

});

결과 : APPLE, BANANA, CHERRY

2. map : 배열 각 요소를 변환하여 새로운 배열을 만든다. 기존 배열은 변경되지 않고 새로운 배열을 반환한다.

const numbers = [1, 2, 3, 4];

const doubled = numbers.map(function(num) {

return num \* 2; // 각 요소를 2배로

});

console.log(doubled); // [2, 4, 6, 8]

3. filter : 배열의 각 요소를 조건에 따라 필터링하여 새로운 배열을 만든다. 조건을 만족하는 요소만 반환한다.

const numbers = [1, 2, 3, 4, 5, 6];

const evenNumbers = numbers.filter(function(num) {

return num % 2 === 0; // 조건 : 짝수

});

console.log(evenNumbers); // [2, 4, 6]

또는

const words = ['apple', 'banana', 'cherry', 'kiwi'];

const longWords = words.filter(function(word) {

return word.length > 5; // 길이가 5 이상인 단어만

});

console.log(longWords); // ['banana', 'cherry']

4. reduce : 배열의 모든 요소를 누적처리하여 하나의 값으로 만든다.

초기값을 설정할 수 있고, 초기값이 없으면 배열의 첫 번째 요소가 초기값으로 사용된다.

예)

const numbers = [1, 2, 3, 4];

const sum = numbers.reduce(function(accumulator, current) {

return accumulator + current; // 누적 합계

}, 0);

console.log(sum); // 10

// 초기값: 0

accumulator : 누적값

current : 현재 요소 값

5. some : 배열의 요소 중 하나라도 조건을 만족하면 true 를 반환한다.

모든 요소가 조건을 만족하지 않으면 false

예)

const numbers = [1, 2, 3, 4, 5];

// 배열에 짝수가 있는지 확인

const hasEven = numbers.some(function(num) {

return num % 2 === 0; // 짝수 조건

});

console.log(hasEven); // true (2, 4가 짝수)

rest parameter

함수에 전달된 인수들의 나머지를 배열로 모을 때 사용.

점 세개를 사용하여 정의한다.

여러개의 인수를 받아야하거나, 특정 인수를 제외한 나머지를 다룰 때 유용하다.

예)

function myFunction(a, b, ...rest) {

console.log(a); // 첫 번째 인수

console.log(b); // 두 번째 인수

console.log(rest); // 나머지 인수들 (배열 형태)

}

myFunction(1, 2, 3, 4, 5);

// 출력:

// 1

// 2

// [3, 4, 5]

\*\*\*\*

. . . rest 가 항상 마지막에 와야한다!

Arrow function (화살표 함수)

예 : const add = (a, b) => {

return a+b;

} ;

중괄호, return 생략 가능

const add = (a,b) => a + b ;

1급 객체

: 다른 객체와 동일하게 취급되는 개체를 말한다.

함수가 1급 객체로 간주된다.

특징)

1. 변수나 데이터 구조에 할당할 수 있다.

함수는 변수에 할당되거나, 배열이나 객체와 같은 데이터 구조에 저장될 수 있다.

배열에 함수를 저장하는것이 가능하다는 뜻!!

예)

const greet = function(name) {

return `Hello, ${name}`;

};

const funcs = [greet, console.log];

console.log(funcs[0]('Bob')); // Hello, Bob

funcs[0] = greet

따라서,

console.log(funcs[0](‘Bob')); 는

greet(‘Bob’) 과 같다

Object의 개념

키-값 쌍을 통해 데이터를 저장하고 관리하는 컨테이너.

key : 고유 식별자 역할

값 : 데이터.

object 생성방법)

1. 객체 리터럴

const person = {

name: 'Alice', // 키: 값

age: 25,

greet: function() {

console.log('Hello!');

}

};

console.log(person.name); // Alice

person.greet(); // Hello!

2. object 생성자

const obj = new Object();

obj.name = 'Bob';

obj.age = 30;

console.log(obj); // { name: 'Bob', age: 30 }

오브젝트 메서드 : 객체에 정의된 함수

const calculator = {

add: (a, b) => a + b,

subtract: (a, b) => a - b

};

console.log(calculator.add(2, 3)); // 5

console.log(calculator.subtract(5, 2)); // 3