**자바스크립트 사용법**

* <script>~</script> : script 태그 안에 Javascript 코드 사용
* <script src=”파일경로” /> : script 태그 안에 src 속성에 자바스크립트 파일 경로 삽입

<body>

    <button onclick="alert('Are you sure?')">확인</button>

</body>

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <link rel="stylesheet" href="style.css">

    <title>first App</title>

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

</head>

**console.log**

console.log("Hello World");

console.log('Hello@@@');

console.log('hello@@@','world','javascript');

**브라우저 확인 및 크롬 개발자모드의 활용**

VS code 에서 alt+b 눌러서 크롬 브라우저 연동  
크롬 브라우저에서 F12(개발자모드) 를 눌러서 콘솔탭에서 확인 가능

**변수 선언**

let str='Hello Javascript';

let number=30;

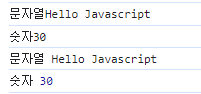
console.log("문자열" + str);

console.log("숫자" + number);

console.log("문자열" ,str);

console.log("숫자" , number);

console.log 에 + 와 , 로 변수를 연결했을 때 출력 유의



**var 와 let**

* 1. ‘var’ 와 ‘let’ 은 JavaScript 에서 변수를 선언하는 데 사용되는 두 가지 키워드
  2. 변수스코프의 차이가 있음
     1. var : 함수 스코프
     2. let : 블록 스코프 { }
  3. 호이스팅 (변수가 선언되기 전에도 참조할 수 있다는 개념)
     1. var : 호이스팅 허용
     2. let : 호이스팅 허용하지만 선언되기 전에 변수를 사용하려고 하면 에러가 발생 (이건 호이스팅을 허용하지 않는것 아닌가)
  4. 재선언
     1. var : 같은 스코프 내에서 여러번 재선언 가능
     2. let : 같은 스코프 내에서 변수를 재선언할 수 없음
  5. 전역 객체 속성
     1. var : 전역 범위에서 선언하면 전역 객체의 속성이 된다. 브라우저 환경에서는 ‘window’ 객체의 속성이 된다.
     2. let : 전역 객체의 속성이 되지 않음
  6. 일반적으로 블록 스코프와 변수 재선언을 허용하지 않는 let 을 사용하는 것이 더 현대적

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **사용 목적** | **스코프** | **호이스팅** | **재선언** | **전역객체속성** |
| **var** | 변수 | 함수 | 허용 | 가능 | 전역 |
| **let** | 변수 | 블록 | 허용(엄격) | 불가능 | 전역아님 |

**let 과 const**

JavaScript에서 let와 const는 변수를 선언하는 데 사용되는 키워드로, 변수를 만들고 값을 저장하는 데 사용됩니다. 이 두 키워드는 var와 비교하여 몇 가지 중요한 차이점이 있습니다.

1. let 변수:
   1. let는 블록 범위(block-scoped) 변수를 선언합니다. 이는 변수가 선언된 블록 내에서만 유효하며, 블록을 벗어나면 변수에 접근할 수 없습니다.
   2. let으로 선언한 변수는 재선언이 허용되지 않습니다. 같은 스코프 내에서 동일한 변수명을 다시 선언하면 에러가 발생합니다.
   3. let 변수는 호이스팅(hoisting)됩니다. 즉, 변수가 선언되기 전에 변수를 참조하면 undefined가 됩니다.
2. const 상수:
   1. const는 블록 범위 상수를 선언합니다. 한 번 할당된 값을 변경할 수 없으며, 상수는 항상 선언과 동시에 초기화되어야 합니다.
   2. const로 선언한 변수는 재할당이 허용되지 않지만 변수가 참조하는 객체 또는 배열의 내용은 수정할 수 있습니다. 즉, 객체 또는 배열 자체는 불변이 아니지만 다른 값을 할당할 수 없습니다.
   3. const 변수도 호이스팅됩니다.

let와 const 중 어떤 것을 사용할지는 변수의 변경 가능성과 스코프에 따라 달라집니다. let은 변경 가능한 변수로 사용되며, const는 상수 값 또는 변경할 수 없는 변수로 사용됩니다. 이것은 코드의 가독성과 안정성을 향상시키는 데 도움이 됩니다.

0 함수와객체

**function**

// function

function sayHello()

{

    alert("Say!! Hello");

    console.log("normal result");

}

function Info()

{

    alert("Information!!!");

}

<button onclick="Info();">Information</button>

**event**

 <button onclick="sayHello();">인사</button>

<body>

    <button onclick="alert('Are you sure?')">확인</button>

</body>

**object type**

// object type

let obj = {

    number:30, // property

    sayHello: function() {

        console.log('obj->Hello1');

        console.log('obj->Hello2');

        console.log('obj->Hello3');

    },

};

console.log("object type", obj);

**HTML 요소 가져오기**

**document.get\*\*\* 함수를 통해 가져오기**

1. document.getElementById : ID 를 참조해서 HTML 요소 가져오기. ID 는 문서에 고유해야 하기 때문에 Element (단수) 를 사용한다.
2. document.getElementsByClassName : class 를 참조해서 HTML 요소 가져오기
3. document.getElementsByTagName : HTML 태그명으로 요소를 가져오기

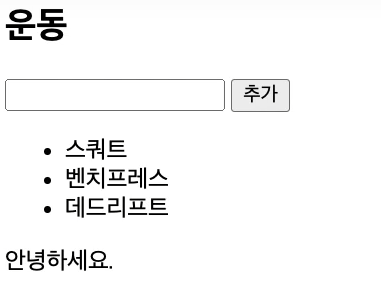
**query\*\*\* 를 이용해 가져오기**

document.query\*\*\* 함수들은 CSS 선택자를 사용해 요소를 찾는 메서드.

get 메서드와 같은 방법도 유용하지만, ID나 class, tag 같은 한 가지 조건이 아니라 다른 요소와의 관계를 사용해서 원하는 요소를 찾는 훨씬 더 강력하고 범용적인 메서드도 있다.

1. document.querySelector(CSS Selector) – 지정된 선택자에 일치하는 문서 내 첫 번째 요소를 반환. 하나를 조회
2. document.querySelectorAll – 지정된 선택자에 일치하는 요소 목록을 반환 . 여러개를 조회

**예제1) 아래 화면을 만드시오.**



<code 1\_selector.html>

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <script src="1\_selector.js"></script>

    <title>exercise</title>

</head>

<body>

    <section>

        <h2 id=”title”>운동</h2>

        <form name="myForm">

            <input type="text">

            <button type="button">추가</button>

        </form>

        <ul class="list">

            <li class="item">스쿼트</li>

            <li class="item">벤치프레스</li>

            <li class="item">데드리프트</li>

        </ul>

        <p class="hello">안녕하세요</p>

    </section>

</body>

</html>

[1\_selector.html]

* form
  1. action 속성
     1. form tag 는 기본 생성이 action 속성과 함께 생성
     2. action 속성은 웹양식(form)이 제출될 때 해당 양식 데이터를 전송할 서버측 스크립트 또는 리소스의 URL 을 지정하는 데 사용. 양식 데이터를 전달할 목적지
     3. action 속성 값을 비워둘 경우 양식 데이터를 현재 페이지로 다시 전송 (자체 제출 또는 현재 페이지로 제출 이라고 함)
  2. name 속성
     1. ‘name’ 속성은 해당 양식을 식별하고 접근할 때 사용됨
        + JavaScript 파일에서 다음과 같이 참조할 수 있음.   
          name=”myForm” 일 경우  
            
          // 양식을 JavaScript로 참조  
          var form = document.forms.myForm;  
            
          // 양식 필드에서 데이터 가져오기  
          var username = form.elements.username.value;
  3. section 을 사용했음
  4. p 를 사용했음

**예제 2) 1\_selector.html 파일에서 id 가 title 인 html 요소를 가져오시오.**

var title = document.getElementById("title");

console.log(title);

이 코드는 null 을 보여줄 수 있음. HTML 코드에서 참조할 스크립트에 **defer** 속성을 부여해야 함

**defer**

1. HTML 의 <script> 태그에서 사용할 수 있는 속성 중 하나
2. 차단하지 않음 : 사용자가 페이지를 볼 때 스크립트 다운로드가 백그라운드에서 진행되므로 초기 페이지 로드 속도가 향상
3. 순서 유지 : defer 속성을 가진 스크립트는 웹 페이지의 파싱이 완료된 후에 실행됨. 스크립트가 페이지의 요소들을 조작하기 위해 사용할 때 해당 요소들이 이미 페이지에 존재하고 준비된 상태로 실행됨
4. 여러 스크립트와 함께 사용 가능 : 실행 순서 보장. 나열된 순서대로 실행
5. 지원되지 않는 브라우저 : HTML5 이전 버전을 사용하는 오래된 브라우저에서는 지원되지 않을 수 있음
6. 페이지의 초기 로딩 속도를 향상시키고 스크립트 실행 순서를 제어하는 데 유용한 방법. 페이지에서 스크립트가 중요한 역할을 하는 경우, ‘defer’ 를 사용하여 페이지 성능을 최적화할 수 있음

**예제 3) JavaScript 를 이용해서 위 페이지에서 “운동” 을 “헬스3대 운동” 으로 변경**

var title = document.getElementById("title");

console.log(title);

title.textContent="헬스3대 운동";

**예제 4) document.get\*\*\* 과 document.querySelector\* 을 사용한 예제**

console.log("-----This is 1\_selector.js");

// 1 get method

var title = document.getElementById("title");

console.log(title);

title.textContent="헬스3대 운동";

let items = document.getElementsByClassName("item");

items[1].textContent = "신제품 벤치프레스";

document.getElementsByTagName("li")[0].textContent = "신제품 스쿼트";

// DOM. HTML 요소 쿼리

console.log("------------DOM----------");

let \_title = document.querySelector("title");

console.log(\_title);

let h2 = document.querySelector("#title");

console.log(h2);

let itemAll = document.querySelectorAll(".item");

console.log(itemAll[0]);

console.log(itemAll[1]);

console.log(itemAll[2]);

**HTML 요소 조작하기**

h2.textContent = "<span>Exercise!!!!</span>";

h2.innerHTML = "<span>Exercise!!!!</span>";

document.querySelector("body").textContent = "Exercise!!";

**예제) JavaScript 를 사용해서 아래 코드의 input 의 placehodler 를 “좋아하는 운동을 입력해주세요” 로 바꾸시오**

 <form name="myForm">

            <input type="text" placeholder="입력을 해주세요">

            <button type="button">추가</button>

        </form>

[코드 3]

let input = document.querySelector("input");

input.setAttribute('placeholder', "좋아하는 운동을 입력해주세요");

**예제) JavaScript 를 이용해서 코드3 의 placeholder 의 값을 지우시오**

input.removeAttribute('placeholder');

**예제)**

let input = document.querySelector("input");

input.setAttribute('placeholder', "좋아하는 운동을 입력해주세요");

input.removeAttribute('placeholder');

input.setAttribute('required', '');

input.removeAttribute('required');

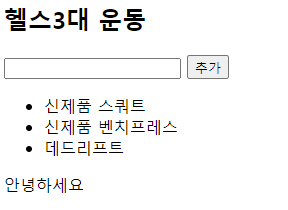
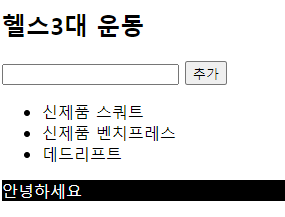
**HTML 요소 스타일링**

1. **요소 프로퍼티 직접 수정**  
   style 프로퍼티를 사용하여 직접 수정

const btn = document.getElementById(‘button’);  
btn.style.color = ‘white’;  
btn.style.backgroundColor = ‘black’;

1. **CSS 클래스 이용**  
   classList 객체를 사용하여 class 를 수정  
     
   const btn = document.getElementById(‘button’);  
   btn.classList.add(‘dark’);  
   btn.classList.remove(‘dark’);

**예제) [그림 1] 을 [그림 2] 로 JavaScript 를 이용해서 바꾸시오.**

[그림 1] [그림 2]

1. style 을 사용하는 방법

// HTML 요소 스타일링

// 1. style 을 사용하는 방법

let \_hello = document.querySelector(".hello");

\_hello.style.color='white';

\_hello.style.backgroundColor='black';

1. classList 를 사용하는 방법

.dark

{

    color : white;

    background-color : black;

}

.light

{

    color : red;

    background-color: yellow;

}

[style.css]

classList 를 사용하지 않고, style.css 를 적용해서 HTML 에 바로 적용한 코드

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <link rel="stylesheet" href="1\_style.css">

    <script src="1\_selector.js" defer ></script>

    <title>exercise</title>

</head>

<body>

    <section>

        <h2 id="title">운동</h2>

        <form name="myForm">

            <input type="text" placeholder="입력을 해주세요">

            <button type="button">추가</button>

        </form>

        <ul class="list">

            <li class="item">스쿼트</li>

            <li class="item">벤치프레스</li>

            <li class="item">데드리프트</li>

        </ul>

        <p class="hello dark">안녕하세요</p>

    </section>

</body>

</html>

[index.html]

JavaScript 의 classList 를 사용해서 add 하는 코드

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <link rel="stylesheet" href="1\_style.css">

    <script src="1\_selector.js" defer ></script>

    <title>exercise</title>

</head>

<body>

    <section>

        <h2 id="title">운동</h2>

        <form name="myForm">

            <input type="text" placeholder="입력을 해주세요">

            <button type="button">추가</button>

        </form>

        <ul class="list">

            <li class="item">스쿼트</li>

            <li class="item">벤치프레스</li>

            <li class="item">데드리프트</li>

        </ul>

        <p class="hello">안녕하세요</p>

    </section>

</body>

</html>

[1\_selector.html]

// 2. CSS 를 활용해 classList 를 사용하는 방법

let \_hello = document.querySelector(".hello");

\_hello.classList.add('dark');

[1\_selector.js]

여러개의 클래스를 add 하는 경우 , 로 구분해서 추가

\_hello.classList.add('dark', 'yellow');

요소를 삭제하고 싶을 경우 remove

\_hello.classList.remove('dark','yellow');

새로운 HTML 요소 만들기

1. createElement – 지정한 tagName 의 HTML 요소를 생성

let element = document.createElement('p');

1. appendChild – 부모의 마지막 자식 요소로 추가

let element = document.createElement('p');

element.textContent = "Added Letter P";

document.querySelector('li').appendChild(element);

1. insertBefore – 추가할 HTML 요소의 앞에 추가

insertBefore(추가할 요소, 추가할 위치)

let element = document.createElement('li');

let insertLocation = document.querySelector(".item");

element.textContent = 'Added Equipment';

document.querySelector('ul').insertBefore(element, insertLocation );

여기서 document.querySelector(‘ul’). 으로 하지 않으면 동작하지 않음.  
부모를 반드시 명시해야 하는것 같음.

예제) 버튼을 누르면 알림이 뜨도록 해보자

<button type="button" onclick="alert('추가하시겠습니까?')">추가</button>

버튼에 이벤트의 동적 추가

1. 이벤트를 동적으로 생성할 button 을 가져온다
   1. getElementById();
2. addEventListener 로 이벤트 추가
   1. 추가할 이벤트는 On제거후 사용. onclick --> click
3. 이벤트 발생시 동작할 함수 작성

예제) <button type=”button”>추가</button>

이 코드를 클릭하면 “추가하시겠습니까?” 팝업이 뜨도록, 이벤트를 동적으로 생성하는 JS 코드를 작성하시오.

// let dynamicButton = document.getElementById('button');

let dynamicButton = document.querySelector('button');

dynamicButton.addEventListener('click',function()

{

    alert('Add??????');

});

여기서 function() 대신 alert(‘Add????’) 를 직접 넣으면 잘 동작하지 않음.

버튼을 누르지 않아도 브라우저가 갱신될때 바로 alert 가 실행됨.

예제) input 에 입력한 글자가 list 로 추가되도록 만들어 보자

let dynamicLi = document.createElement('li');

let \_button = document.querySelector('button');

\_button.addEventListener('click', function()

{

    let \_text = document.querySelector('input');

    dynamicLi.textContent = \_text.value;

    document.querySelector('ul').appendChild(dynamicLi);

    \_text.value="";

})

그런데, 이렇게 하면, 하나만 추가되고 다시 추가하려고 하면 기존에 추가한게 지워진다.

let \_button = document.querySelector('button');

\_button.addEventListener('click', function()

{

    let \_text = document.querySelector('input');

    let dynamicLi = document.createElement('li');

    dynamicLi.textContent = \_text.value;

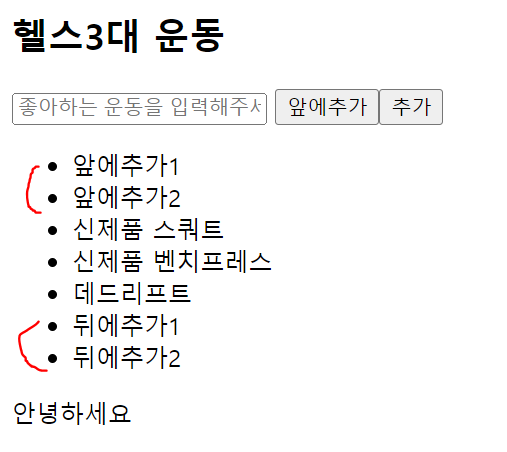
    document.querySelector('ul').appendChild(dynamicLi);

    \_text.value="";

})

이렇게 해야 한다 .

예제) “앞에추가” 버튼을 누르면 앞에 추가되게 해보자.



let \_button = document.querySelector('button');

\_button.addEventListener('click', function()

{

    let \_text = document.querySelector('input');

    let dynamicLi = document.createElement('li');

    dynamicLi.textContent = \_text.value;

    document.querySelector('ul').appendChild(dynamicLi);

    \_text.value="";

})

let \_addedButton = document.createElement('button');

\_addedButton.textContent = '앞에추가';

\_addedButton.type = 'button';

let \_addedButtonLocation = document.querySelector("button");

document.querySelector("form").insertBefore(\_addedButton, \_addedButtonLocation);

\_addedButton.addEventListener('click', function(){

    let element1 = document.createElement('li');

    let location1 = document.querySelector('.item');

    let textvalue = document.querySelector('input');

    element1.textContent = textvalue.value;

    document.querySelector('ul').insertBefore(element1, location1);

    textvalue.value = "";

})

* 새로 생성한 button 의 타입을 정하지 않으면 기본적으로 “submit” 이 되서 새로고침되면서 추가가 되지 않는다.
* 실수로 클래스를 참조할때 .item 과 같이 id 를 참조하는 코드를 넣으면 아무것도 찾지 못해서 결국 </ul> 태그 앞에 생성될 것이다.