DOM 조작하기 1

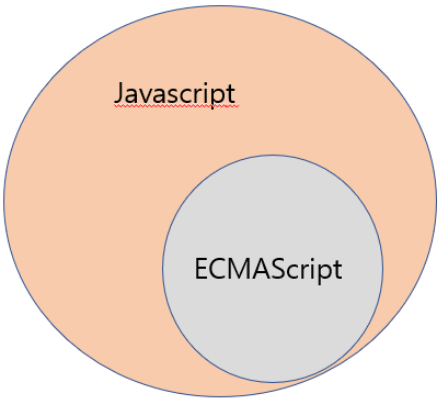
자료	Javascrtip
구분	Javascript
과목	

JavaScript 란?

JavaScript는 웹 브라우저에서 동작하는 유일한 프로그래밍 언어이다.
다른 언어들은 주로 애플리케이션 개발을 위한 도구로 설계 되었지만,
JavaScript는 처음부터 웹페이지의 보조 기능을 처리하기 위한 제한적인 용도로 만들어졌다.
하지만 지금 JavaScript는 프론트엔드와 백엔드 영역을 모두 아우를 수 있는
강력한 개발 언어로 성장했다.

우리는 나중에 Node.js 라는 JavaScript 실행 환경을 설치하게 될 것이다.
원래 JavaScript는 브라우저 엔진에서만 실행 됐었지만,
Node.js를 통해 브라우저 외의 환경인 서버나 데스크탑 등에서도
JavaScript를 사용할 수 있게 된 것이다.

이번에는 ECMAScript 가 무엇인지도 살펴보자.
JavaScript 표준 문법을 ECMAScript라고 할 수 있다.
ECMAScript는 JavaScript에서 값, 타입, 객체, 함수 등의 표준 문법 등을 규정한다.



JavaScript는 ECMAScript보다 상위 개념이다.
ECMAScript는 일반적인 프로그래밍 언어로서의 뼈대를 이루는 표준 문법이고,

JavaScript는 이 ECMAScript에 더해서 브라우저가 제공하는 Web API들(DOM, BOM 등)을 포함하는 개념이다.

즉, JavaScript는 **ECMAScript + Web API**를 아우르는 더 넓은 개념이라고 보면 된다.

JavaScript의 구성

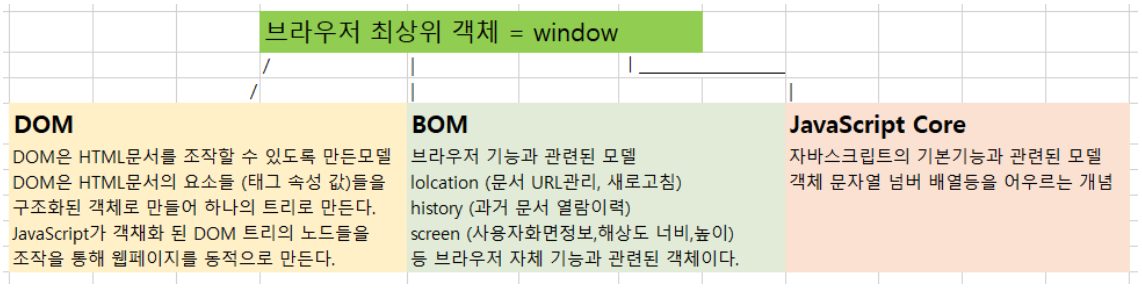
JavaScript는 브라우저를 하나의 객체로 간주한다. 이 객체를 `window`라고 부른다.

즉, 브라우저에서 JavaScript를 실행하면

모든 전역 변수와 함수들은 `window` 객체의 속성이나 메서드로 작동하게 된다.

`window` 객체는 크게 세 가지로 나눌 수 있다.

바로 **DOM**, **BOM**, 그리고 **JavaScript Core** 이다.



- **DOM (Document Object Model)**: HTML 문서를 객체로 표현한 구조로, JavaScript가 문서의 내용과 구조를 동적으로 조작할 수 있게 해준다.

  예시: `document.getElementById("title").textContent = "Hello!";`
- **BOM (Browser Object Model)**: 브라우저 창과 관련된 객체들(window, history, location 등)을 다루는 모델이다.

  예시: `history.back();`
- **JavaScript Core**: DOM이나 BOM과는 별개로, 프로그래밍 언어로서의 순수 JavaScript 기능(변수, 함수, 조건문 등)을 말한다.

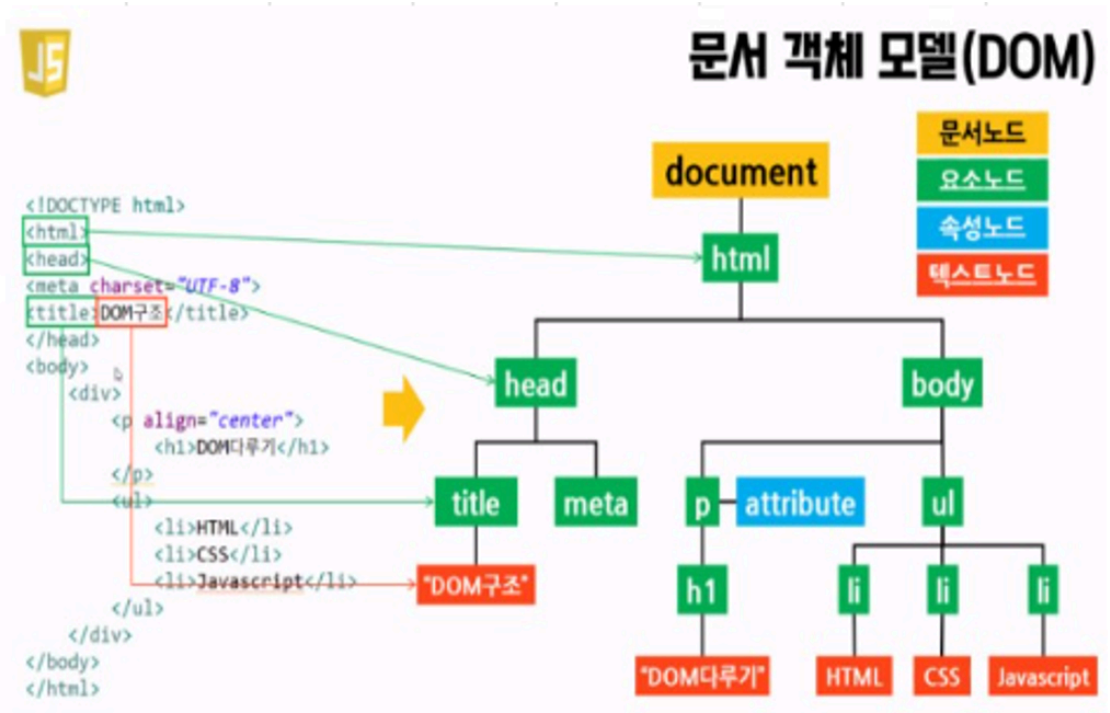
  예시: `let sum = a + b;`

오늘 수업의 핵심은 **DOM을 조작하는 것**이다.

DOM은 JavaScript가 웹페이지에 접근해서 HTML 태그나 속성 같은 요소들을 조작할 수 있도록,

HTML 문서의 요소들을 **트리(tree)** 형태로 표현한 자료구조이다.

이것을 **DOM 트리**라고 부른다.



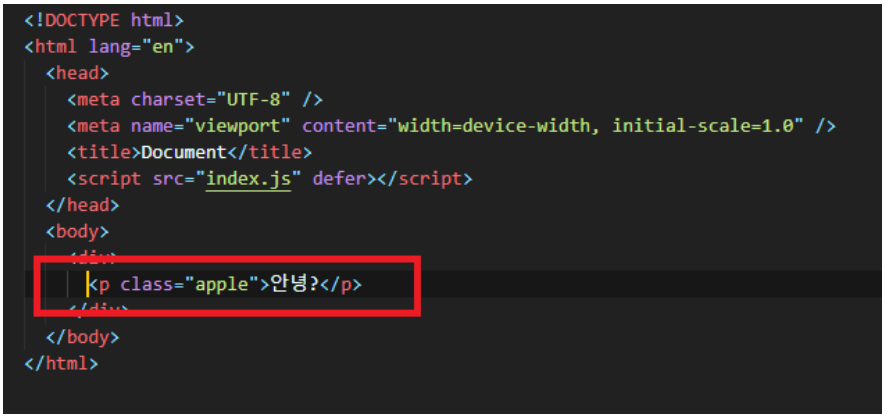
위 그림을 보면 DOM 트리는 부모-자식 관계로 이루어져 있다는 걸 확인할 수 있다.
DOM 트리의 최상단 루트 노드이자 객체를 `document`라고 부른다.

그 아래에 있는 상위 노드는 `<html>`이고, 그 밑에 `<head>`와 `<body>`가 자식 노드가 된다.
우리가 웹페이지를 방문하면, 웹페이지가 화면에 로딩되는 순서는 대략 이렇게 진행된다:

1. 웹 브라우저 실행
2. 브라우저가 웹 문서를 읽음 (`document`)
3. DOM 트리를 생성 → HTML 문서의 모든 요소를 객체 형태로 변환
4. 페이지 로딩 → CSS 스타일 적용 + 자바스크립트를 통한 동적 요소 반영
5. 최종적으로 화면에 표시

이런 식으로 웹페이지가 구성된다고 보면 된다.

DOM 트리를 구성하는 Node를 크게 3종류로 구분할 수 있다.



1. 요소노드 (Element Node)
    a. div 그리고 p와 같은 모든 HTML 태그를 의미하는 노드다
2. 속성노드 (Attribute Node)
    a. class id href 와 같이 HTML 태그의 속성(attribute)을 나타내는 노드다.
3. 텍스트노드 (Text Node)
    a. 안녕? 과 같은 텍스트 내용을 의미하는 노드다.

노드들을 트리의 구조로 나타내면 다음과 같다.

<p> (요소 노드)
├── class="apple" (속성 노드)
└── "안녕?" (텍스트 노드)

우리가 오늘 할 일은 JavaScript를 사용해서

DOM 객체들을 조작하는 것이라고 했다.

DOM 조작을 통해 웹페이지를 동적으로 바꿀 수 있기 때문이다.

DOM 조작의 기본 원리

DOM 조작은 크게 두 단계로 이루어 진다.

1. 요소 선택하기
2. 선택한 요소 조작하기

이 간단한 두 단계만 마스터하면 웹 페이지를 자유자재로 변경할 수 있다.

1. 요소 선택 방법

[사전 셋팅] HTML 문서 body에 아래 코드 추가한다.

```
<div id="container">
  <h1 id="title">과일 목록</h1>
```

```
  <ul id="fruits-list">
    <li class="fruit red">사과</li>
    <li class="fruit yellow">바나나</li>
    <li class="fruit orange">오렌지</li>
  </ul>
  <div id="info-box"></div>
</div>
```

1.1 단일 요소 선택하기

body에 <script> 코드를 추가 한 다음에 script 태그 안에 아래 코드를 넣는다

```
// ID로 요소 선택하기
const title = document.getElementById('title');
console.log(title);

// CSS 선택자로 첫 번째 일치하는 요소 선택하기
const container = document.querySelector('#container');
console.log(container); // <div id="container">...</div>

// 첫 번째 과일 항목 선택하기
const firstFruit = document.querySelector('.fruit');
console.log(firstFruit); // <li class="fruit red">사과</li>
```

1.2 여러 요소 선택하기

```
// 클래스로 여러 요소 선택하기
const fruits = document.getElementsByClassName('fruit');
console.log(fruits); // HTMLCollection(3) [li.fruit.red, li.fruit.yellow, li.fruit.orange]

// CSS 선택자로 여러 요소 선택하기
const fruitItems = document.querySelectorAll('.fruit');
console.log(fruitItems); // NodeList(3) [li.fruit.red, li.fruit.yellow, li.fruit.orange]

// 모든 과일 항목 반복하기
fruitItems.forEach(fruit => {
  console.log(fruit.textContent);
});
// 출력:
// 사과
// 바나나
// 오렌지
```

메서드	반환 값	활용 상황
getElementById	단일 요소	ID가 있는 단일 요소를 빠르게 선택할 때

querySelector	첫 번째 일치 요소	복잡한 CSS 선택자로 요소를 선택할 때
getElementsByClassName	HTMLCollection	동적 컬렉션이 필요할 때 (실시간 업데이트)
querySelectorAll	NodeList	반복 메서드(forEach)를 사용하거나 정적 컬렉션이 필요할 때

처음 JS를 학습 할 때에는 우성적으로 querySelector 그리고 querySelectorAll만

잘 사용하는 것을 연습하자.

2. 요소 조작 방법

요소를 선택한 후에는 다양한 방법으로 조작할 수 있다.

2.1 요소 생성 및 추가하기

```
// 새로운 요소 생성하기
const newFruit = document.createElement('li');
newFruit.textContent = '포도';
newFruit.className = 'fruit purple';

// 요소를 목록의 끝에 추가하기
const fruitsList = document.querySelector('#fruits-list');
fruitsList.appendChild(newFruit);

// 요소를 목록의 시작에 추가하기
const newFruit2 = document.createElement('li');
newFruit2.textContent = '키위';
newFruit2.className = 'fruit green';
fruitsList.prepend(newFruit2);

// 특정 위치에 요소 삽입하기
const newFruit3 = document.createElement('li');
newFruit3.textContent = '망고';
newFruit3.className = 'fruit yellow';
const orangeFruit = document.querySelector('.orange');
fruitsList.insertBefore(newFruit3, orangeFruit);
```

2.2 요소 삭제하기

```
// 요소 직접 삭제하기
const bananaFruit = document.querySelector('.yellow');
bananaFruit.remove();

// 부모를 통해 자식 요소 삭제하기
const redFruit = document.querySelector('.red');
fruitsList.removeChild(redFruit);
```

2.3 요소 내용 변경하기

```
// textContent: 텍스트만 다루기
const title = document.querySelector('#title');
title.textContent = '맛있는 과일 목록';

// innerHTML: HTML 포함하여 내용 설정하기
const infoBox = document.querySelector('#info-box');
infoBox.innerHTML = '<p>이 목록에는 <strong>다양한 과일</strong>이 포함되어 있습니다.</p>';
```

텍스트를 다룰 때 사용하는 속성으로는 innerHTML, innerText 그리고 textContent 세가지가 있다.

innerHTML 속성은 HTML 까지 포함해서 읽고/쓸 수 있다.

그렇다면 textContent와 innerText의 차이도 살펴보자.

2.4 textContent와 innerText의 차이

```
const info = document.querySelector('#info-box');
info.innerHTML='<p>보이는 텍스트 <span style="display: none;">숨겨진 텍스트</span></p>';

console.log(info.textContent); // 출력: 보이는 텍스트 숨겨진 텍스트
console.log(info.innerText);   // 출력: 보이는 텍스트
```

textContent는 HTML 구조와 상관없이 모든 텍스트 노드를 가져온다. 따라서 display:none 이 되어도 무조건 text가 포함이 된다.

반면에 innerText는 실제로 렌더링된 텍스트만 가져온다. 따라거 display:none 요소는 무시하게 되는 것이다.

3. 속성 조작하기

HTML 요소의 속성을 조작하는 것은 웹 페이지의 동적 변경에 중요하다.

3.1 직접 속성 가져오기와 설정하기

```
// 속성 가져오기
const fruitsList = document.querySelector('#fruits-list');
console.log(fruitsList.id); // 출력: fruits-list

// 속성 설정하기
fruitsList.id = 'new-fruits-list';
console.log(fruitsList.id); // 출력: new-fruits-list
```

3.2 getAttribute 또는 setAttribute 사용하기

```
// getAttribute 로 속성의 값을 읽어오기
const orangeFruit = document.querySelector('.orange');
```

```
// setAttribute 로 속성을 새로 만들거나 또는 기존 속성값을 변경할때 사용한다.
orangeFruit.setAttribute('data-origin', '제주도');
console.log(orangeFruit.getAttribute('data-origin')); // 출력: 제주도

// 속성 제거하기
orangeFruit.removeAttribute('data-origin');
```

4. 클래스 조작하기

클래스 조작은 CSS와 연동하여 스타일을 동적으로 변경할 때 유용하다.

```
// 요소 선택
const mangoFruit = document.querySelector('.fruit:nth-child(3)');

// 클래스 추가하기
mangoFruit.classList.add('sweet');
console.log(mangoFruit.className); // 출력: fruit yellow sweet

// 클래스 제거하기
mangoFruit.classList.remove('yellow');
console.log(mangoFruit.className); // 출력: fruit sweet

// 클래스 토글하기 (있으면 제거, 없으면 추가)
mangoFruit.classList.toggle('organic'); // 추가됨
console.log(mangoFruit.className); // 출력: fruit sweet organic

mangoFruit.classList.toggle('organic'); // 제거됨
console.log(mangoFruit.className); // 출력: fruit sweet

// 클래스 포함 여부 확인하기
console.log(mangoFruit.classList.contains('sweet')); // 출력: true
```

5. DOM 메서드 정리표

아래는 DOM조작을 위한 메서드를 정리한 표이다.

눈으로 한번 훑어 보도록 하자.

목적	메서드/속성	설명
요소 선택	getElementById(id)	ID로 단일 요소 선택
	querySelector(selector)	CSS 선택자로 첫 번째 요소 선택
	getElementsByClassName(className)	클래스명으로 요소 컬렉션 선택
	querySelectorAll(selector)	CSS 선택자로 모든 요소 선택
요소 생성/추가	createElement(tagName)	새 HTML 요소 생성
	appendChild(node)	자식 요소 끝에 추가

	prepend(node)	자식 요소 시작에 추가
	insertBefore(newNode, referenceNode)	참조 노드 앞에 삽입
요소 삭제	remove()	요소 자체 삭제
	removeChild(child)	부모로부터 자식 요소 삭제
내용 조작	textContent	요소의 텍스트 내용 (모든 텍스트)
	innerText	요소의 보이는 텍스트 내용
	innerHTML	HTML을 포함한 내용
속성 조작	getAttribute(name)	속성값 가져오기
	setAttribute(name, value)	속성값 설정하기
	removeAttribute(name)	속성 제거하기
클래스 관리	classList.add(className)	클래스 추가
	classList.remove(className)	클래스 제거
	classList.toggle(className)	클래스 토글 (추가/제거)
	classList.contains(className)	클래스 포함 여부 확인

6. [참고] NodeList는 배열이 아니다.

`document.querySelectorAll('p')`를 사용하면, **모든** `p` **태그를 선택**해서 `NodeList`라는 객체가 생성된다.

하지만 이 `NodeList`는 **배열**이 아니어서, 배열에서 제공하는 `map`, `filter`, `reduce` 같은 메서드는 사용할 수 없다는 점을 반드시 기억하자.

6.1 NodeList의 특징

1. **NodeList**는 **배열이 아닌 객체**로, 선택된 요소들을 나열한 유사 배열(like array)이다.
2. `forEach()` 메서드는 사용 가능:

   예시:

```
const pTags = document.querySelectorAll('p');
pTags.forEach(p => {
  console.log(p.textContent);  // 각 p 태그의 텍스트를 출력
});
```

1. **배열 메서드 사용 불가**:

   `NodeList`에서는 `map`, `filter`, `reduce` 같은 메서드를 사용할 수 없다.

```
const pTags = document.querySelectorAll('p');
pTags.map(tag => tag.textContent);  // 에러 발생! map은 사용할 수 없음
```

6.3 NodeList를 배열로 변환해보기

- 하지만 `NodeList`는 배열은 다르지만, `NodeList`를 **배열로 변환**해서 사용할 수 있다.

  배열 메서드를 사용하고 싶다면 `Array.from()` 을 이용해 `NodeList`를 배열로 변환할 수 있다.

  예시:

  ```
  const pTags = document.querySelectorAll('p');
  const pArray = Array.from(pTags);  // NodeList를 배열로 변환
  pArray.map(tag => tag.textContent);  // 이제 배열 메서드 사용 가능하지만 우리 아직 map학습 안함
  ```

끝