

JavaScript란?

- 웹 브라우저에서 동적인 기능을 구현할 수 있는 프로그래밍 언어입니다.
- 스크립트 언어 : 이미 존재하는 소프트웨어를 제어하기 위한 용도로 쓰이는 언어
- HTML과 CSS가 내용과 디자인을 담당한다면, JavaScript는 동작을 담당합니다.

전역 객체(Global Object)

웹 브라우저 환경에서 매우 핵심적인 정보 저장과 역할을 담당합니다. 다음 3가지 객체는 브라우저 기반의 JavaScript 프로그램에서 UI 및 환경에 접근하고 제어할 수 있도록 돕습니다.

1. screen 객체

- 브라우저가 실행 중인 “사용자의 물리적인 화면(디스플레이)에 대한 정보”를 담고 있음
- 자주 사용되지는 않음

속성	설명
<code>screen.width</code>	화면의 전체 너비 (픽셀 단위)
<code>screen.height</code>	화면의 전체 높이
<code>screen.availWidth</code>	OS의 작업 표시줄 등을 제외한 사용 가능한 너비
<code>screen.availHeight</code>	사용 가능한 높이
<code>screen.pixelDepth</code>	픽셀당 색상 깊이 (ex. 24, 32)
<code>screen.orientation</code>	화면의 방향 (portrait, landscape 등)

2. window 객체

- [브라우저 창\(window\)](#) 자체를 나타냄
- window 객체의 구성 요소인 전역 함수나 전역 변수들은 자주 사용됨.

주요 역할

- 타이머 제어: `window.setTimeout`, `window.setInterval`
- 팝업 제어: `window.open()`, `window.close()`
- 브라우저 제어: `window.location`, `window.history`
- 이벤트 리스너 추가: `window.hrefaddEventListener`

💡 대부분의 경우 `window.` 는 생략하고 사용함.

속성/메서드	설명
<code>window.innerWidth</code> , <code>innerHeight</code>	뷰포트(스크롤바 제외 영역)의 너비/높이
<code>window.location</code>	현재 URL 정보 (리다이렉트, 페이지 이동 가능)
<code>window.alert()</code>	경고창 표시 <code>/confirm()/prompt()</code>
<code>window.setTimeout()</code>	일정 시간 후 코드 실행
<code>window.localStorage</code>	로컬 저장소

3. document 객체 (DOM)

1. `document` 가 HTML DOM() =>
2. HTML DOM() =>
3. CSS
4. `.(head script body - *)`

- 현재 브라우저에 로드된 HTML 문서를 의미
- DOM(Document Object Model)을 통해 HTML 요소를 탐색하거나 조작할 수 있음

주요 기능

- HTML 요소 선택 및 조작
- 이벤트 처리
- 동적 HTML 생성

메서드	설명
<code>document.getElementById()</code>	ID로 요소 찾기
<code>document.querySelector()</code>	CSS 셀렉터 기반 요소 찾기
<code>document.createElement()</code>	새 요소 생성
<code>document.body, document.head</code>	문서의 본문/헤드 요소 접근
<code>document.title</code>	문서의 제목 읽기/수정
<code>document.addEventListener()</code>	문서 레벨의 이벤트 처리


자바스크립트 문법

1. 변수 (값을 저장하는 공간)

```
let name = "Alice"; // 바뀔 수 있는 값
const age = 25;      // 바뀌지 않는 값
```

- **let**: 값을 나중에 바꿀 수 있음
 - **const**: 한 번 정하면 바꿀 수 없음
 - **var**: 옛날 방식 (지양)
-

2. 자료형 (데이터 종류)

 자료형	예시
문자열 (String)	"hello"
숫자 (Number)	123, 3.14
불리언 (Boolean)	true, false
null	값이 없음
undefined	정의되지 않음
객체 (Object)	{ name: "Tom" }
객체 (Object)	[1, 2, 3] (배열 (Array))

```
let isOnline = true;
let user = { name: "Tom", age: 30 };
let fruits = ["apple", "banana"];
```

3. 연산자

- 산술: `+`, `-`, `*`, `/`, `%`
- 비교: `==`, `===`, `!=`, `!==`, `<`, `>`, `<=`, `>=`
- 논리: `&&`, `||`, `!`

```
let x = 10;
let y = 5;
console.log(x + y);           // 15
console.log(x > y);           // true
console.log(x === "10");      // false
```

4. 함수 (Function)

- 어떤 작업을 묶어서 재사용할 수 있음

```
function greet(name) {
  console.log("Hello, " + name);
}
```

```
greet("Alice"); // Hello, Alice
```

- 화살표 함수 (짧은 문법)

```
const greet = (name) => {
  console.log("Hi " + name);
};
```

5. 배열과 객체

- 배열(**Array**): 여러 값을 순서대로 저장
- 객체(**Object**): 이름(**key**)과 값(**value**)을 쌍으로 저장

```
let colors = ["red", "green", "blue"];  
console.log(colors[1]); // green
```

```
let user = {  
  name: "Tom",  
  age: 30  
};  
console.log(user.name); // Tom
```

6. 콘솔과 디버깅

- `console.log()`를 이용해 값이나 흐름을 확인할 수 있음

```
let a = 5;  
console.log("a의 값:", a);
```

DOM

1. DOM (Document Object Model)

- 브라우저가 HTML 문서를 객체 구조로 표현한 것.
- JavaScript로 HTML 요소를 선택하고 조작할 수 있게 함.
- 예: `document.querySelector('div')`는 `<div>` 요소를 JS에서 객체로 가져옴.

2. DOM 스타일 속성 조작

- DOM 요소의 스타일 변경은 `element.style.속성명`으로 가능

```
const box = document.querySelector(".box");  
box.style.color = "red"; // 글자색을 빨간색으로 변경
```

- 이처럼 **HTML** 구조는 그대로 두고 **CSS**만 자바스크립트로 제어할 수 있음.

3. 스타일 속성명: `background-color` → `backgroundColor`

- JavaScript에서는 `background-color`처럼 ****하이픈(-)****이 들어간 CSS 속성을 `camelCase`로 바꿔 써야 함:

```
element.style.backgroundColor = "blue";
```

이유: `background-color`는 자바스크립트에서 유효하지 않은 식별자 (예약어처럼 해석될 수 있음)

4. 예약어와 식별자

- JavaScript에는 사용할 수 없는 ****예약어(reserved word)****가 있음
(ex. `class`, `return`, `default` 등)

5. 기본 문법 학습의 중요성

- DOM 조작, 이벤트 처리, 함수 사용 등은 기본 문법에 대한 이해가 전제
 - 무작정 외우기보다는 개념 + 연습 중심으로 학습하는 것이 중요
-

JavaScript 이벤트 처리

✓ 1. 이벤트란?

- 사용자 또는 브라우저가 발생시키는 동작 신호
 - 예: 클릭, 입력, 마우스 이동, 키보드 입력, 페이지 로드 등
버튼 클릭 → **click** 이벤트
키보드 입력 → **keydown** 이벤트
-

✓ 2. 이벤트 처리(Event Handling)

이벤트를 감지하고, 반응하는 동작을 **JS**로 작성하는 것

방법 1: **HTML**에 직접 쓰기

```
<button onclick="alert('클릭됨!')">버튼</button>
```

방법 2: **JavaScript**에서 처리하기 (권장)

```
const btn = document.querySelector("button");  
btn.addEventListener("click", function () {  
    alert("클릭됨!");  
});
```

✓ 3. 주요 이벤트 종류

이벤트 이름	설명
click	마우스 클릭
mouseover	마우스 오버
mouseout	마우스 벗어남
keydown	키보드 키 누름
keyup	키보드 키 땀
change	input, select 값 변경

input	입력창의 값이 실시간으로 변함
submit	폼 제출
load	페이지 또는 이미지 로딩 완료

✓ 4. **addEventListener()** 기본 문법

```
element.addEventListener("이벤트이름", 함수);
```

예시:

```
const input = document.querySelector("#name");

input.addEventListener("input", function () {
  console.log("입력값:", input.value);
});
```

✓ 5. 이벤트 객체 (**event** 또는 **e**)

이벤트 핸들러 함수에 전달되는 특별한 객체로, 이벤트의 세부 정보를 담고 있음.

```
btn.addEventListener("click", function (event) {
  console.log("이벤트 타입:", event.type);
  console.log("클릭한 위치:", event.clientX,
    event.clientY);
});
```

✓ 6. 폼 이벤트: **submit** 막기

html -----

```
<form id="myForm">
  <input type="text" required />
  <button type="submit">제출</button>
</form>
```

js -----

```
const form = document.getElementById("myForm");

form.addEventListener("submit", function (e) {
  e.preventDefault(); // 기본 제출 막기
  alert("폼 제출됨!");
});
```

✅ 7. 이벤트 위임 (Delegation) - 고급 개념

부모 요소에 이벤트를 달고, 자식 요소의 이벤트를 감지
(동적으로 생성된 요소에도 적용 가능)

```
document.getElementById("list").addEventListener("click",
  function (e) {
    if (e.target.tagName === "LI") {
      e.target.style.color = "red";
    }
  });
```

🧠 자바스크립트 함수 형식과 화살표 함수

✅ 1. 함수 선언 방식 종류

📌 (1) 함수 선언식 (Function Declaration)

```
function greet(name) {
  return "Hello, " + name;
}

greet("Tom"); // Hello, Tom
```

- 호이스팅(끌어올림) 가능 → 함수 정의 코드 보다 위쪽에서 호출해도 됨
-

(2) 함수 표현식 (Function Expression)

```
const greet = function(name) {  
  return "Hello, " + name;  
};
```

```
greet("Jane"); // Hello, Jane
```

- 변수에 함수를 익명으로 저장
 - 호이스팅되지 않음
-

(3) 화살표 함수 (Arrow Function)

```
const greet = (name) => {  
  return "Hello, " + name;  
};
```

```
greet("Alex"); // Hello, Alex
```

- 더 짧은 문법으로 함수 작성 가능
 - **function** 키워드를 생략
-

2. 화살표 함수의 축약 문법

```
// 매개변수 1개 → 괄호 생략 가능  
const square = x => x * x;
```

```
// 매개변수 2개 이상 → 괄호 필요  
const add = (a, b) => a + b;
```

```
// 여러 줄이면 중괄호 + return 필요
const greet = name => {
  console.log("Hi");
  return "Hello, " + name;
};
```

⚠ 3. 화살표 함수의 주의점

제한 사항	설명
this	화살표 함수는 자신만의 this 를 가지지 않음
arguments	arguments 객체 없음
생성자 함수로 사용 불가	new TestFunc() → TestFunc 가 화살표 함수인 경우 new 키워드 사용 못함.

🔍 예시: **this** 차이

```
const user = {
  name: "Tom",
  greet: function () {
    console.log(this.name);
  },
};

const userArrow = {
  name: "Jane",
  greet: () => {
    console.log(this.name); // ❌ 예상과 다르게 undefined
  },
};

user.greet(); // Tom
userArrow.greet(); // undefined (this는 외부 컨텍스트)
```



4. 함수 선언 방식 비교 요약

구분	함수 선언식	함수 표현식	화살표 함수
문법	<code>function</code> 사용	변수에 익명 함수 대입	<code>=></code> 사용
호이스팅	O	X	X
<code>this</code> 바인딩	함수 호출 시 동적	함수 호출 시 동적	정적 (부모 스코프 고정)
<code>arguments</code> 사용	가능	가능	✗ 불가
생성자 사용	가능	가능	✗ 불가
용도	일반 함수 정의	콜백, 변수 저장 함수	콜백, 간단한 함수

✅ 언제 화살표 함수를 쓰면 좋을까?

- ✓ 콜백 함수나 짧은 함수 표현이 필요할 때
 - ✓ `setTimeout` 등 콜백 안에서 `this` 유지가 필요할 때 유용
 - ✓ 복잡한 기능보다 간단한 동작 처리가 목적일 때
-

콜백 함수 (Callback Function)

1. 콜백 함수란?

"다른 함수에 인자로 전달되는 함수"
→ 특정 작업이 끝난 뒤에 "나중에 호출되는 함수"

즉, 어떤 함수가 다른 함수 내부에서 실행되도록 전달된 함수입니다.

2. 왜 콜백 함수를 쓸까?

- 자바스크립트는 버튼을 클릭하거나, 데이터를 가져올 때, 그 일이 끝난 뒤 "무엇을 할지" 를 함수로 전달합니다.
-

3. 콜백 함수 기본 예제

```
function greeting(name) {  
  alert("안녕하세요, " + name + "님!");  
}  
  
function processUserInput(callback) {  
  const name = prompt("당신의 이름은?");  
  callback(name); // 콜백 함수 실행  
}  
  
processUserInput(greeting);
```

- greeting 은 processUserInput 함수에 인자로 전달된 함수
 - processUserInput이 언제 어떻게 실행할지 결정하고 나서 호출
-

4. 예제: **setTimeout** (비동기 콜백)

```
setTimeout(function () {  
    console.log("3초 후 실행됨!");  
}, 3000);
```

- 3초가 지난 후 콜백 함수가 실행됨
 - `setTimeout`은 지정된 시간이 지나면 콜백 함수를 실행함
-

5. 배열 메서드에서의 콜백 함수

forEach, map, filter, reduce 등에서 사용

```
const numbers = [1, 2, 3];  
numbers.forEach(function (num) {  
    console.log(num * 2);  
});
```

- `forEach`는 배열의 각 요소마다 콜백 함수를 자동으로 실행
 - `num`은 `numbers`에 저장된 각 배열 요소이며 순서대로 가져온 값을 `num`에 저장하여 함수를 각각 실행함
-

6. 콜백 지옥 (Callback Hell)

콜백을 중첩해서 사용할 경우 코드가 너무 복잡해지는 현상입니다.
