자바스크립트(JavaScript)란?

자바스크립트(JavaScript)는 객체(object) 기반의 스크립트 언어입니다.
HTML로는 웹의 내용을 작성하고,
CSS로는 웹을 디자인하며,
자바스크립트로는 웹의 동작을 구현할 수 있습니다.

자바스크립트는 주로 웹 브라우저에서 사용되나, Node.js와 같은 프레임워크를 사용하면 서버 측 프로그래밍에서도 사용할 수 있습니다.
현재 컴퓨터나 스마트폰 등에 포함된 대부분의 웹 브라우저에는 자바스크립트 인터프리터가 내장되어 있습니다.

자바스크립트의 특징

1) 자바스크립트는 객체 기반의 스크립트 언어입니다.
2) 자바스크립트는 동적이며, 타입을 명시할 필요가 없는 인터프리터 언어입니다.
3) 자바스크립트는 객체 지향형 프로그래밍과 함수형 프로그래밍을 모두 표현할 수 있습니다.

객체, 속성, 메소드의 개념 이해

자바스크립트는 객체 지향적 언어입니다. 생활주변에서 볼 수 있는 사람, 동물, 자전거, 꽃 등 모든 사물이 객체입니다.

자바스크립트에서 객체(object)란 웹브라우저를 포함한 웹 문서의 모든 구성요소를 말합니다. 즉, 웹브라우저의 상태표시줄, 스크롤바, 웹문서 자체, 레이어, 하이퍼링크, 이미지, 폼버튼 등등 대부분의 구성요소를 객체라고 합니다.
객체들은 또 다른 하위 객체들을 가지고 있을 수 있습니다. 이런 이유로 자바스크립트의 객체구조를 계층적구조라고 말할 수 있습니다.

모든 객체는 속성(Property)을 가집니다.
window 객체는 frames(프레임), name(윈도우 이름), location(위치) 등의 속성을 가지며 이미지 객체는 가로크기, 세로크기등의 속성을 가집니다.

document.title="연습"; (객체.속성="속성값");

객체와 속성, 객체와 메소드를 연결 할 때에는 (.) 으로 연결합니다

메소드 이해하기

메소드(method)는 객체의 동작과 관련된 것입니다.

「토끼」라는 객체를 예를 들면,「빛깔이 희다」,「앞다리가 짧다」 등은 속성에 관한 사항이고,「뛰어간다」,「풀을 뜯어 먹는다」 등의 행동은 메소드에 해당합니다.

자바스크립트에서 open() 메소드는 팝업 윈도우를 열어주고, write() 메소드는 문자열을 출력하며 alert() 메소드는 경고창을 열어 줍니다.

HTML 요소에 쓰기 – innerHTML.

HTML 출력 – document.write().

alert box(경고창) - window.alert().

browser console에 쓰기 – console.log().

- 기본 명령어(메소드)

1) document.write()

document.write() 메소드(특별한 행동을 하는 자바스크립트 함수; 동사에 해당)에 들어가는 문자열에는 태그도 사용이 가능합니다.

한가지 명심할 것은 문자열은 반드시 따옴표 사이에 들러싸여 있어야 되고, 따옴표 안에 또 따옴표를 넣어야 될 경우에는 반드시 홑따옴표(' ')를 사용해야 됩니다. 그렇지 않으면 브라우저는 에러를 발생시키게 됩니다.

따옴표로 둘러싸여진 문자열은 아무리 길이가 길어도 절대 엔터키를 쳐서 줄바꿈은 하지 않도록 합니다.

HTML 문서가 로드된 후 document.write()를 사용하면 기존 HTML이 모두 삭제됩니다.

document.write() 메서드는 테스트용으로만 사용해야 합니다.

2) alert()

window.alert() 메서드는 경고 상자를 사용하여 데이터를 표시할 수 있습니다.

window키워드는 생략 가능합니다. 예) alert();

alert함수는 메시지와 OK 버튼만을 가진 다이얼로그 박스를 보여주는 함수로 사용자의 요구를 받을 필요가 없는 메시지와 경우에 사용합니다.

3) confirm()

confirm함수는 메시지와 OK/Cancel버튼을 포함한 다이얼로그 박스를 보여주는 함수로, 사용자로부터 응답을 듣고 싶을 때 사용합니다. 사용자가 OK버튼을 누르면 true를, Cancel 버튼을 누르면 false를 반환합니다.

4) prompt()

prompt 함수는 메시지와 입력 필드를 가진 다이얼로그 박스를 보여주는 함수로 사용자로부터 숫자나 문자열을 입력받고자 할 때 사용합니다.

5) console.log()

디버깅을 위해 console.log()브라우저에서 메서드를 호출하여 데이터를 표시할 수 있습니다.

6) window.print()

window.print()브라우저에서 메서드를 호출하여 현재 창의 내용을 인쇄할 수 있습니다.

스크립트는 <body>, 또는 <head>HTML 페이지의 섹션 또는 둘 다에 배치할 수 있습니다.
JavaScript 파일의 파일 확장자는 .js 입니다.
외부 스크립트를 사용하려면 <script> 태그의 src(source) 속성에 스크립트 파일 이름을 입력합니다.

HTML ID요소에 액세스하기 위해 JavaScript는 document.getElementById(id)메서드를 사용할 수 있습니다.

```
<!DOCTYPE html>
<html>
<head>
  <script>
     function myFunction() {
       document.getElementById("demo").innerHTML = "Paragraph changed.";
  }
</script>
</head>
<body>
  <h2>Demo JavaScript in Head</h2>
  <p id="demo">A Paragraph</p>
  <button type="button" onclick="myFunction()">Try it</button>
</body>
</html>
```

```
<!DOCTYPE html>
<html>
<body>
  <h2>Demo JavaScript in Body</h2>
  <p id="demo">A Paragraph</p>
  <button type="button" onclick="myFunction()">Try it</button>
  <script>
  function myFunction() {
    document.getElementById("demo").innerHTML = "Paragraph changed.";
  }
  </script>
</body>
</html>
```

세미콜론은 JavaScript 문을 구분합니다.
각 실행 가능한 문의 끝에 세미콜론을 추가합니다.

JavaScript는 여러 공백을 무시합니다. 스크립트에 공백을 추가하여 가독성을 높일 수 있습니다.

```
let person = "Hege";
let person="Hege";
```

좋은 방법은 연산자(= + - * /) 주위에 공백을 넣는 것입니다.

```
let x = y + z;
```

최고의 가독성을 위해 프로그래머는 종종 80자보다 긴 코드 라인을 피하고 싶어합니다.
JavaScript 문이 한 줄에 맞지 않는 경우 가장 좋은 위치는 연산자 다음입니다.

예)

```
document.getElementById("demo").innerHTML =
    "Hello Dolly!";
```

JavaScript 문은 중괄호 {...} 안에 있는 코드 블록으로 그룹화할 수 있습니다.
코드 블록의 목적은 함께 실행할 명령문을 정의하는 것입니다.
블록으로 함께 그룹화된 명령문을 찾을 수 있는 곳은 JavaScript 함수입니다.

```
function myFunction() {
  document.getElementById("demo1").innerHTML = "Hello Dolly!";
  document.getElementById("demo2").innerHTML = "How are you?";
}
```

자바스크립트 키워드
JavaScript 문은 수행할 JavaScript 작업을 식별하는 키워드로 시작하는 경우가 많습니다.
예약어 참조에는 모든 JavaScript 키워드가 나열되어 있습니다.
JavaScript 키워드는 예약어입니다. 예약어는 변수 이름으로 사용할 수 없습니다.

Keyword	설명
var	변수 선언
let	블록 변수 선언
const	블록 상수 선언
if	조건에서 실행할 명령문 블록을 표시합니다.
switch	다른 경우에 실행할 명령문 블록을 표시합니다.
for	루프에서 실행할 명령문 블록을 표시합니다.
function	함수 선언
return	기능 종료
try	명령문 블록에 대한 오류 처리를 구현합니다.

JavaScript Syntax

JavaScript Values(값)

JavaScript 구문은 두 가지 유형의 값을 정의합니다.
Fixed values(고정 값) - Literals
Variable values(변수 값) - Variables

JavaScript Literals
고정 값에 대한 가장 중요한 두 가지 구문 규칙은 다음과 같습니다.

1. 숫자는 소수를 포함하거나 포함하지 않고 작성됩니다.
2. 문자열은 큰따옴표나 작은따옴표로 묶인 텍스트입니다.

JavaScript Variables

프로그래밍 언어에서 변수는 데이터 값을 저장하는 데 사용됩니다.
JavaScript는 키워드 var, let, const를 사용하여 변수를 선언합니다.
var키워드는 1995년부터 2015년까지 모든 JavaScript 코드에 사용되었습니다.
let및 키워드 const는 2015년에 JavaScript에 추가되었습니다.
이전 브라우저에서 코드를 실행하려면 var를 사용

JavaScript Let

let으로 정의된 변수는 재선언할 수 없습니다.
let으로 정의된 변수는 사용 전에 선언되어야 합니다.
let으로 정의된 변수에는 블록 범위가 있습니다.
({ } 블록 내부에 선언된 변수는 블록 외부에서 액세스할 수 없습니다.)
let으로 정의된 변수는 실수로 변수를 다시 선언할 수 없습니다.
예)

```
let x = "John Doe";
let x = 0;
// SyntaxError: 'x' has already been declared
{
  let x = 2;
}
// x can NOT be used here
```

{ } 블록 내부에 선언된 변수는 블록 외부에서 액세스할 수 없습니다.
let또는 const.로 선언된 변수는 다시 선언할 수 없습니다

```
let carName = "Volvo";
let carName;
```

==>이것은 작동하지 않습니다.

JavaScript var

프로그램의 모든 위치에서 JavaScript 변수를 다시 선언하는 var은 허용됩니다.
선언된 JavaScript 변수를 다시 선언하면 var값이 손실되지 않습니다.

```
var carName = "Volvo";
var carName;
```

변수 carName는 다음 명령문을 실행한 후에도 여전히 "Volvo" 값을 가집니다.

JavaScript Const

const로 정의된 변수는 재선언할 수 없습니다.
const로 정의된 변수는 재할당할 수 없습니다.
const로 정의된 변수에는 블록 범위가 있습니다.
JavaScript const변수는 선언될 때 값을 할당해야 합니다.

`const PI = 3.14159265359;`	O
`const PI;` `PI = 3.14159265359;`	X

```
const price1 = 5;
const price2 = 6;
```
두 개의 변수 price1및 price2 는 const 키워드로 선언됩니다.
이들은 상수 값이며 변경할 수 없습니다.
다른 범위 또는 다른 블록에서 const를 사용하여 변수를 다시 선언하는 것은 허용됩니다.

```
const x = 2;       // Allowed
// Here x is 2
{
  const x = 3;   // Allowed
// Here x is 3
}

{
  const x = 4;   // Allowed
// Here x is 4
}
```

등호(=) 는 변수에 값을 할당 하는데 사용됩니다.

x라는 변수에 값 6이 할당됩니다.

```
let x;
x = 6;
```

자바스크립트 주석
JavaScript 주석은 JavaScript 코드를 설명하고 더 읽기 쉽게 만드는 데 사용할 수 있습니다.
한 줄 주석 - 한 줄 주석은 // 로 시작합니다.
여러 줄 주석 – 여러 줄 주석은 /* 로 시작 하고 */ 로 끝납니다.

이중 슬래시 //또는 /*및 사이의 코드는 주석*/ 으로 처리됩니다 .
주석은 무시되며 실행되지 않습니다.

자바스크립트 식별자/이름

모든 JavaScript 변수는 고유한 이름으로 식별되어야 합니다.
이러한 고유한 이름을 식별자라고 합니다.
식별자는 JavaScript 이름입니다.
식별자는 변수, 키워드, 함수의 이름을 지정하는 데 사용됩니다.
한번에 1개의 데이터만 저장 가능합니다.
새로운 데이터가 입력되면 기존의 값은 '삭제'됩니다.
예약어(reserved word)는 자바스크립트에서 특별한 용도를 가진 키워드를 의미하며, 변수명으로 사용할 수 없다.

JavaScript 이름은 다음으로 시작해야 합니다.

1. 문자(AZ 또는 az)
2. 달러 기호($)
3. 또는 밑줄(_)
4. 후속 문자는 문자, 숫자, 밑줄 또는 달러 기호일 수 있습니다.
5. 숫자는 이름의 첫 문자로 사용할 수 없습니다.
6. JavaScript는 대소문자를 구분합니다

모든 JavaScript 식별자는 대소문자를 구분합니다.

변수 lastName및 lastname는 두 가지 다른 변수입니다.
하이픈(-)은 JavaScript에서 허용되지 않습니다. 뺄셈을 위해 예약되어 있습니다.

식별자는 짧은 이름(예: x 및 y)이거나 더 설명적인 이름(연령, 합계, totalVolume)일 수 있습니다.

변수, 함수, 객체 등을 선언할 때는 변수명, 함수명, 객체명과 같이 이름이 필요합니다.
이러한 이름에 일정한 규칙을 부여하면 코드에 대한 가독성을 높일 수 있습니다.

표기법	설명
camelCase(카멜)표기법	첫 번째 단어의 첫 문자는 소문자, 두 번째 단어 이후부터는 첫 문자만 대문자로 표시한다. ->userAge, createElement()
Pascal(파스칼)표기법	각 단어의 첫 글자를 대문자로 표시한다. -> UserAge, SpeedDirection()
underscore(언더스코어)	각 단어를 언더바(_)로 이어준다. -->user_age, time_process()

변수명과 함수명은 Camel표기법으로, 객체는 Pascal 표기법으로 일정한 규칙을 만들어 사용하고, 변수는 명사, 함수는 동사로 표현하는 것이 좋습니다.

- 변수에 저장 가능한 데이터의 종류

1) 문자형 데이터(String)

큰따옴표 또는 작은따옴표로 둘러싸인 데이터로 문자

```
var userName = "하민지";
var num = "1000";
```

2). 숫자형 데이터(Number)

```
var num = 100000;
```

3) 논리형 데이터(Boolean)

참, 거짓으로 정의되는 데이터

```
var result = true;
var result = false;
```

4) 널형 데이터(Null) 값이 없음

```
var result = null;
```

HTML DOM Document getElementById() --> 지정된 ID를 가진 요소를 가져온다.

예)

```
const myElement = document.getElementById("demo");
myElement.style.color = "red";
```

HTML DOM Document getElementsByClassName() --> 지정된 클래스 이름을 가진 요소를 가져온다.

예)

```
<div class="example">
  <p>This is a paragraph.</p>
</div>
<br>
<div class="example color">
  <p>This is a paragraph.</p>
</div>
<br>
<div class="example color">
  <p>This is a paragraph.</p>
</div>
<script>
const collection = document.getElementsByClassName("example color");
collection[0].style.backgroundColor = "red";
</script>
```

classList속성은 요소의 CSS 클래스 이름을 반환합니다.
classList 속성 및 메서드

Name	Description
add()	목록에 하나 이상의 토큰을 추가합니다.
contains()	목록에 클래스가 포함되어 있으면 true를 반환합니다.
entries()	목록에서 키/값 쌍이 있는 Iterator를 반환합니다.
forEach()	목록의 각 토큰에 대한 콜백 함수를 실행합니다.
item()	지정된 인덱스의 토큰을 반환합니다.
keys()	목록에 키가 있는 Iterator를 반환합니다.
length	목록의 토큰 수를 반환합니다.
remove()	목록에서 하나 이상의 토큰을 제거합니다.
replace()	목록의 토큰을 대체합니다.
supports()	토큰이 속성이 지원하는 토큰 중 하나이면 true를 반환합니다.
toggle()	목록의 토큰 간 전환
value	토큰 목록을 문자열로 반환
values()	목록의 값을 가진 Iterator를 반환합니다.

HTML DOM Document getElementsByName() --> 지정된 이름을 가진 요소를 가져온다.
예)

```
<p>First Name: <input name="fname" type="text" value="Michael"></p>
<p>First Name: <input name="fname" type="text" value="Doug"></p>
<script>
let elements = document.getElementsByName("fname");
document.getElementById("demo").innerHTML = elements[0].tagName;
</script>
```

HTML DOM Document getElementsByTagName() --> 지정된 태그이름을 가진 요소를 가져온다.
예)

```
<ul>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ul>
<script>
const collection = document.getElementsByTagName("li");
document.getElementById("demo").innerHTML = collection[1].innerHTML;
</script>
```

HTML DOM Document querySelector() -- > CSS 선택기와 일치하는 첫 번째 요소를 반환합니다.
예1) class="example"에서 첫 번째 <p> 요소를 가져옵니다.

```
<h2 class="example">A heading</h2>
<p class="example">A paragraph.</p>

<script>
document.querySelector("p.example").style.backgroundColor = "red";
</script>
```

예2)

```
<p>Change the text of the element with id="demo":</p>
<p id="demo">This is a p element with id="demo".</p>
<script>
document.querySelector("#demo").innerHTML = "Hello World!";
</script>
```

예3) 부모가 <div인 첫 번째 <p> 요소를 선택합니다> 요소.

```
<div>
  <h3>H3 element</h3>
  <p>I am a p element, my parent is a div element.</p>
</div>

<div>
  <h3>H3 element</h3>
  <p>I am a p element, my parent is also a div element.</p>
</div>
<button onclick="myFunction()">Try it</button>
<script>
function myFunction() {
  var x = document.querySelector("div > p");
  x.style.backgroundColor = "red";
}
```

예4) "target" 속성이 있는 첫 번째 <a> 요소를 선택합니다.

```
<a href="https://www.w3schools.com">w3schools.com</a>
<a href="http://www.disney.com" target="_blank">disney.com</a>
<a href="http://www.wikipedia.org" target="_top">wikipedia.org</a>
<script>
document.querySelector("a[target]").style.border = "10px solid red";
</script>
```

예5) 첫 번째 <h3> 또는 첫 번째 <h4>를 선택합니다.

```
<h3>A h3 element</h3>
<h4>A h4 element</h4>
<script>
document.querySelector("h3, h4").style.backgroundColor = "red";
</script>
```

HTML DOM Document querySelectorAll() --> CSS 선택기와 일치하는 모든 요소를 반환합니다.

예)

```
<h2 class="example">A heading</h2>
<p class="example">A paragraph.</p>
<script>
const nodeList = document.querySelectorAll(".example");
for (let i = 0; i < nodeList.length; i++) {
  nodeList[i].style.backgroundColor = "red";
}
</script>
```

JavaScript 연산자의 유형

다양한 유형의 JavaScript 연산자가 있습니다.

산술 연산자
할당 연산자 =
비교 연산자
논리 연산자
조건 연산자
유형 연산자

JavaScript 산술 연산자

산술 연산자 는 숫자에 대한 산술을 수행하는 데 사용됩니다.

Operator	Description
+	더하기
-	빼기
*	곱하기
**	지수
/	나누기
%	나머지
++	Increment(증가)
--	Decrement(감소)

JavaScript 할당 연산자

할당 연산자는 JavaScript 변수에 값을 할당합니다.

더하기 할당 연산자 ( ) 는 +=변수에 값을 추가합니다.

Operator	Example	Same As
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y
**=	x **= y	x = x ** y

JavaScript 문자열 추가

+연산자를 사용하여 문자열을 추가(연결)할 수도 있습니다.

```
let text1 = "John";
let text2 = "Doe";
let text3 = text1 + " " + text2;
```

+=할당 연산자를 사용하여 문자열을 추가(연결)할 수도 있습니다.

```
let text1 = "What a very ";
text1 += "nice day";
```

두 개의 숫자를 더하면 합계가 반환되지만 숫자와 문자열을 더하면 문자열이 반환됩니다.

JavaScript 비교 연산자

Operator	Description
==	equal to(동일)
===	equal value and equal type(동일한 값 및 동일한 유형)
!=	not equal(같지 않다)
!==	not equal value or not equal type(같지 않은 값 또는 같지 않은 유형)
>	greater than(~보다 큰)
<	less than(미만)
>=	greater than or equal to(크거나 같음)
<=	less than or equal to(이하)
?	ternary operator(삼항 연산자)

JavaScript 논리 연산자

Operator	Description
&&	logical and(논리곱)
||	logical or(논리합)
!	logical not(논리부정)

JavaScript 유형 연산자

Operator	Description
typeof	Returns the type of a variable (변수의 유형을 반환합니다.)
instanceof	Returns true if an object is an instance of an object type (객체가 객체 유형의 인스턴스인 경우 true를 반환합니다.)

문자열을 숫자로 변환
전역 메서드 Number()는 변수(또는 값)를 숫자로 변환합니다.

숫자 문자열(예: "3.14")은 숫자(예: 3.14)로 변환됩니다.
Number("3.14")
빈 문자열(예: "")은 0으로 변환됩니다.
Number(" ")
숫자가 아닌 문자열(예: "John")은 NaN(Not a Number)로 변환됩니다.
Number("John")

Method	Description
Number()	인수에서 변환된 숫자를 반환합니다.
parseFloat()	문자열을 구문 분석하고 부동 소수점 숫자를 반환합니다.
parseInt()	문자열을 구문 분석하고 정수를 반환합니다.

단항 + 연산자
단항 + 연산자를 사용하여 다음을 수행할 수 있습니다. 변수를 숫자로 변환

```
let y = "5";      // y is a string
let x = + y;      // x is a number
document.getElementById("demo").innerHTML = typeof y + "<br>" + typeof x;
```

```
let y = "John";   // y is a string
let x = + y;      // x is a number (NaN)
etElementById("demo").innerHTML = typeof y + "<br>" + typeof x;
```

만약 변수는 변환 할 수 없으며 여전히 숫자가되지만 값 (숫자가 아님)이됩니다.NaN

숫자를 문자열로 변환-->String()

전역 메서드는 숫자를 문자열로 변환할 수 있습니다.
모든 유형의 숫자, 리터럴, 변수 또는 표현식에 사용할 수 있습니다.

```
let x = 123;
document.getElementById("demo").innerHTML =
  String(x) + "<br>" +
  String(123) + "<br>" +
  String(100 + 23);
```

toString() 메서드는 숫자를 문자열로 변환합니다.

```
<p id="demo"></p>
let x = 123;
document.getElementById("demo").innerHTML =
  x.toString() + "<br>" +
(123).toString() + "<br>" +
(100 + 23).toString();
```

Method	Description
toExponential()	지수 표기법을 사용하여 반올림되고 작성된 숫자가 포함된 문자열을 반환합니다
toFixed()	숫자를 반올림하고 지정된 소수점 이하 자릿수로 쓴 문자열을 반환합니다.
toPrecision()	지정된 길이로 쓰여진 숫자가 포함된 문자열을 반환합니다.

Date methods

Method	Description
getDate()	날짜를 숫자로 가져오기(1-31)
getDay()	요일을 숫자(0-6)로 가져오기
getFullYear()	네 자리 연도(yyyy) 가져오기
getHours()	시간 가져오기(0-23)
getMilliseconds()	밀리초 가져오기(0-999)
getMinutes()	분 가져오기(0-59)
getMonth()	월 가져오기(0-11)
getSeconds()	초 가져오기(0-59)
getTime()	시간 가져오기(1970년 1월 1일 이후 밀리초)

부울을 숫자로 변환

```
Number(false)     // returns 0
Number(true)      // returns 1
```

부울을 문자열로 변환

```
String(false)     // returns "false"
String(true)      // returns "true"
```

자동 형식 변환

자바 스크립트가 "잘못된"데이터 유형에서 작동하려고하면 값을 "올바른" 형식으로 변환합니다.

```
5 + null    // returns 5         because null is converted to 0
"5" + null  // returns "5null"   because null is converted to "null"
"5" + 2     // returns "52"      because 2 is converted to "2"
"5" - 2     // returns 3         because "5" is converted to 5
"5" * "2"   // returns 10        because "5" and "2" are converted to 5 and 2
```

search() 메서드는 표현식을 사용하여 일치 항목을 검색하고 일치 위치를 반환합니다.
대소문자 구분하지 않는다.

```
let text = "Visit W3Schools!";
let n = text.search("W3Schools");
document.getElementById("demo").innerHTML = n;
```

replace() 메서드는 패턴이 교체된 수정된 문자열을 반환합니다.

```
<button onclick="myFunction()">Try it</button>
<p id="demo">Please visit Microsoft!</p>
<script>
function myFunction() {
  let text = document.getElementById("demo").innerHTML;
  document.getElementById("demo").innerHTML =
  text.replace("Microsoft","W3Schools");
}
</script>
```

JavaScript 연산자 우선 순위
연산자 우선 순위는 산술 식에서 연산이 수행되는 순서를 설명합니다.

곱셈(*)과 나눗셈(/)은 덧셈(+)과 뺄셈(-)보다 우선 순위가 높습니다.
우선 순위가 같은 작업(* 및 / 등)은 왼쪽에서 오른쪽으로 계산됩니다.
let x = 100 + 50 * 3;-->450
let x = 100 / 50 * 3;--> 6

JavaScript Objects

개체에는 많은 값이 포함될 수 있습니다.
JavaScript 개체는 속성 및 메서드라고 하는 명명된 값의 컨테이너입니다.

```
const car = {type:"Fiat", model:"500", color:"white"};
```

```
<p id="demo"></p>
<script>
const person = {
  firstName: "John",
  lastName: "Doe",
  age: 50,
  eyeColor: "blue"
};

document.getElementById("demo").innerHTML =
person.firstName + " is " + person.age + " years old.";
</script>
```

개체는 메소드를 가질 수도 있습니다.
메서드는 개체에 대해 수행할 수 있는 작업입니다.
메서드는 함수 정의로 속성에 저장됩니다.

```
const person = {
  firstName: "John",
  lastName : "Doe",
  id       : 5566,
  fullName : function() {
    return this.firstName + " " + this.lastName;
  }
};
```

```
<p id="demo"></p>
<script>
const person = {};
person.firstName = "John";
person.lastName = "Doe";
person.age = 50;
person.eyeColor = "blue";

document.getElementById("demo").innerHTML =
person.firstName + " is " + person.age + " years old.";
</script>
```

```
<p id="demo"></p>
<script>
const person = new Object();
person.firstName = "John";
person.lastName = "Doe";
person.age = 50;
person.eyeColor = "blue";

document.getElementById("demo").innerHTML =
person.firstName + " is " + person.age + " years old.";
</script>
```

JavaScript Arrays

배열은 둘 이상의 값을 보유할 수 있는 특수 변수입니다.
배열은 단일 이름으로 많은 값을 보유할 수 있으며 색인 번호를 참조하여 값에 액세스할 수 있습니다.
const 키워드로 배열을 선언하는 것이 일반적입니다.
배열 리터럴을 사용하는 것이 JavaScript 배열을 만드는 가장 쉬운 방법입니다.
Javascript에서 배열 리터럴은 표현식 목록으로, 각 표현식은 한 쌍의 대괄호 '[ ] '로 묶인 배열 요소를 나타냅니다.

Syntax:

```
const array_name = [item1, item2, ...];
const points = new Array();
const points = [];
```

```
const cars = ["Saab", "Volvo", "BMW"];
let car1 = "Saab";
let car2 = "Volvo";
let car3 = "BMW";
```

```
const cars = [
  "Saab",
  "Volvo",
  "BMW"
];
```

```
const cars = [];
cars[0]= "Saab";
cars[1]= "Volvo";
cars[2]= "BMW";
```

```
const cars = new Array("Saab", "Volvo", "BMW");
```

자바스크립트에는 내장 배열 생성자 new Array()가 있습니다.
단순성, 가독성 및 실행 속도를 위해 배열 리터럴 방법을 사용하십시오.

색인 번호를 참조하여 배열 요소에 액세스합니다.
배열 인덱스는 0부터 시작합니다.

[0]은 첫 번째 요소입니다. [1]은 두 번째 요소입니다.

```
const cars = ["Saab", "Volvo", "BMW"];
let car = cars[0];
```

배열 이름을 참조하여 전체 배열에 액세스할 수 있습니다.

```
<p id="demo"></p>

<script>
const cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars;
</script>
```

자바스크립트 typeof

1. 자바 스크립트에는 값을 포함 할 수 있는 5가지 데이터 유형이 있습니다.
   - string
   - number
   - boolean
   - object
   - function
2. 객체에는 6가지 유형이 있습니다.
   - Object
   - Date
   - Array
   - String
   - Number
   - Boolean

3. 값을 포함 할 수없는 2 가지 데이터 유형 :
   - null
   - undefined

```
typeof "John"                 // Returns "string"
typeof 3.14                   // Returns "number"
typeof NaN                    // Returns "number"
typeof false                  // Returns "boolean"
typeof [1,2,3,4]              // Returns "object"
typeof {name:'John', age:34}  // Returns "object"
typeof new Date()             // Returns "object"
typeof function () {}         // Returns "function"
typeof myCar                  // Returns "undefined" *
typeof null                   // Returns "object"
```

- NaN의 데이터 유형은 숫자입니다.
- 배열의 데이터 유형은 object입니다.
- 날짜의 데이터 유형은 객체입니다.
- null의 데이터 유형은 객체입니다.
- 정의되지 않은 변수의 데이터 형식 undefined. *
- 값이 할당되지 않은 변수의 데이터 형식 undefined

typeof 연산자는 다음 기본 형식 중 하나를 반환할 수 있습니다.

- string
- number
- boolean
- undefined
- function
- object

typeof 연산자는 개체, 배열 및 null에 대해 "object"를 반환합니다.

```
const fruits = ["Banana", "Orange", "Apple"];
let type = typeof fruits;
```

instanceof 연산자는 개체가 지정된 개체의 인스턴스인 경우 true를 반환합니다.

```
const cars = ["Saab", "Volvo", "BMW"];

document.write(
(cars instanceof Array) + "<br>" +
(cars instanceof Object) + "<br>" +
(cars instanceof String) + "<br>" +
(cars instanceof Number));
```

void연산자

```
<a href="javascript:void(0);">Useless link</a>
```

배열을 문자열로 변환

자바스크립트 toString()메서드는 배열을 쉼표로 구분된 배열 값의 문자열로 변환합니다.

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.write(fruits.toString());
```

join()메서드는 모든 배열 요소를 문자열로 조인합니다.

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.write(fruits.join(" * "));
```

pop()메서드는 배열에서 마지막 요소를 제거합니다.

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.pop();
document.write(fruits);
```

push()메서드는 배열에 새 요소를 추가합니다.

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.push("Kiwi");
document.write(fruits);
```

shift() 메서드는 배열의 첫 번째 요소를 제거하고 다른 요소를 왼쪽으로 "이동"합니다.

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fruit = fruits.shift();
document.write(fruits);
```

unshift() 메서드는 배열의 시작 부분에 새 요소를 추가합니다.

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.unshift("Lemon");
document.write(fruits);
```

length 속성은 배열의 길이(배열 요소의 수)를 반환합니다.
길이 속성은 항상 가장 높은 배열 인덱스보다 하나 더 큽니다.

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let length = fruits.length;
document.write(fruits);
```

concat()메서드는 병합 (연결)하여 새 배열을 만듭니다.

```
const myGirls = ["Cecilie", "Lone"];
const myBoys = ["Emil", "Tobias", "Linus"];
const myChildren = myGirls.concat(myBoys);
document.write(myChildren);
```

```
const array1 = ["Cecilie", "Lone"];
const array2 = ["Emil", "Tobias", "Linus"];
const array3 = ["Robin", "Morgan"];
const myChildren = array1.concat(array2, array3);
document.write(myChildren);
```

splice() 메서드는 배열에 새 항목을 추가하는데 사용할 수 있습니다.

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.write(fruits);
```

```
fruits.splice(2, 0, "Lemon", "Kiwi");
document.write(fruits);
```

첫 번째 매개 변수 (2)는 새 요소를 추가 (접합)해야하는 위치를 정의합니다.
두 번째 매개 변수(0)는 제거 해야 하는 요소 수를 정의합니다.
나머지 매개 변수 ("Lemon"", "Kiwi"")는추가 할 새 요소를 정의합니다.
splice() 메서드는 삭제된 항목이 있는 배열을 반환합니다.

```
fruits.splice(2, 2, "Lemon", "Kiwi");
```

첫 번째 매개 변수 (2)는 새 요소를 추가 (접합)해야하는 위치를 정의합니다.

두 번째 매개 변수(2)는 제거해야 하는 요소 수를 정의합니다.

slice() 메서드는 배열의 일부를 새 배열로 분할합니다.

```
const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
const citrus = fruits.slice(1, 3);
```

sort() 메서드는 배열을 알파벳순으로 정렬합니다.

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.sort();
```

reverse() 메서드는 배열의 요소를 반대로 바꿉니다.

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.reverse();
```

forEach() 메서드는 각 배열 요소에 대해 한 번씩 함수(콜백 함수)를 호출합니다.
이 함수는 3 개의 인수를 사용합니다.

- 항목 값(value)
- 항목 인덱스(index)
- 배열 자체(array)

```
const numbers = [45, 4, 9, 16, 25];
let txt = "";
numbers.forEach(myFunction);
document.write(txt);
function myFunction(value, index, array) {
  txt += value+ "<br>";
}
```

indexOf() 메서드는 배열에서 요소 값을 검색하고 해당 위치를 반환합니다.
첫 번째 항목의 위치 0, 두 번째 항목의 위치 1 등입니다.

```
const fruits = ["Apple", "Orange", "Apple", "Mango"];
let position = fruits.indexOf("Apple");
array.indexOf(item, start)
```

item	검색할 항목입니다. 필수
start	검색을 시작할 위치. 음수 값은 주어진 위치에서 시작하여 끝에서 세고 끝까지 검색합니다.

Array.indexOf()항목을 찾을 수 없는 경우 -1을 반환합니다.
항목이 두 번 이상 있으면 첫 번째 위치를 반환합니다.

Array.lastIndexOf() 지정된 요소의 마지막 발생 위치를 반환합니다.

```
const fruits = ["Apple", "Orange", "Apple", "Mango"];
let position = fruits.lastIndexOf("Apple")
```

JavaScript에서 배열은 번호가 매겨진 인덱스를 사용합니다.
JavaScript에서 개체는 명명된 인덱스를 사용합니다.

Object.keys() 메서드는 객체의 키가 있는 Array Iterator(배열 반복자) 객체를 반환합니다.

Syntax

```
Object.keys(object)
```

Use Object.keys() on an array:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
const keys = Object.keys(fruits);
```

Use Object.keys() on a string:

```
const fruits = "Banana";
const keys = Object.keys(fruits);
```

Use Object.keys() on an object:

```
const person = {
  firstName: "John",
  lastName: "Doe",
  age: 50,
  eyeColor: "blue"
};
const keys = Object.keys(person);
```

instanceof 연산자는 객체가 주어진 생성자에 의해 생성된 경우 true를 반환합니다.

```
<p id="demo"></p>

<script>
var fruits = ["Banana", "Orange", "Apple"];
document.getElementById("demo").innerHTML = fruits instanceof Array;
</script>
```

자바스크립트 this 키워드
this는 변수가 아닙니다. 키워드입니다. this의 값은 변경할 수 없습니다.

```
const person = {
  firstName: "John",
  lastName: "Doe",
  id: 5566,
  fullName : function() {
    return this.firstName + " " + this.lastName;
  }
};
document.write(person.fullName());
```

자바스크립트 조건문

JavaScript if, else, and else if

조건문은 다양한 조건에 따라 다양한 작업을 수행하는 데 사용됩니다.
if를 사용하여 지정된 조건이 참인 경우 실행할 코드 블록을 지정합니다.
동일한 조건이 거짓인 경우 실행할 코드 블록을 지정하려면 else를 사용하십시오.
첫 번째 조건이 거짓인 경우 else if를 사용하여 테스트할 새 조건을 지정합니다.
실행할 코드의 많은 대체 블록을 지정하려면 스위치를 사용하십시오.
if 문을 사용하여 조건이 참인 경우 실행할 JavaScript 코드 블록을 지정합니다.
else 문을 사용하여 조건이 거짓인 경우 실행할 코드 블록을 지정합니다.

```
if (hour < 18) {
  greeting = "Good day";
}
if (조건) {
  // 조건이 참이면 실행할 코드 블록
} else {
  // 조건이 거짓일 때 실행할 코드 블록
}
```

첫 번째 조건이 거짓이면 else if 문을 사용하여 새 조건을 지정합니다.

```
if (조건1) {
  // 조건1이 참일 경우 실행할 코드 블록
} else if (조건2) {
  // 조건1이 false이고 조건2가 true인 경우 실행될 코드 블록
} else {
  // 조건1이 false이고 조건2가 false인 경우 실행될 코드 블록
}
```

```
if (time < 10) {
  greeting = "Good morning";
} else if (time < 20) {
  greeting = "Good day";
} else {
  greeting = "Good evening";
}
```

JavaScript Switch Statement

switch 문은 다른 조건에 따라 다른 작업을 수행하는 데 사용됩니다.
실행할 많은 코드 블록 중 하나를 선택하려면 switch 문을 사용하십시오.

```
switch(expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}
```

switch 식은 한 번 평가됩니다.
식의 값은 각 경우의 값과 비교됩니다.
일치하는 항목이 있으면 연결된 코드 블록이 실행됩니다.
일치하는 항목이 없으면 기본 코드 블록이 실행됩니다.

```
switch (new Date().getDay()) {
  case 0:
    day = "Sunday";
    break;
  case 1:
    day = "Monday";
    break;
  case 2:
    day = "Tuesday";
    break;
  case 3:
    day = "Wednesday";
    break;
  case 4:
    day = "Thursday";
    break;
```

```
  case 5:
    day = "Friday";
   break;
  case 6:
    day = "Saturday";
}
```

JavaScript For Loop

Loop는 코드 블록을 여러 번 실행할 수 있습니다.
매번 다른 값으로 동일한 코드를 반복해서 실행하려는 경우 편리합니다.

```
text += cars[0] + "<br>";
text += cars[1] + "<br>";
text += cars[2] + "<br>";
text += cars[3] + "<br>";
text += cars[4] + "<br>";
text += cars[5] + "<br>";

for (let i = 0; i <cars.length; i++) {
  text += cars[i] + "<br>";
}
```

놓기 조건 증감

```
for (표현식1; 표현식2; 표현식3) {
  // code block to be executed
}
```

`for (let i = 0; i < arr.length; i++) {`	X
`let n = arr.length;` `for (let i = 0; i < n; i++) {`	O

잘못된 코드는 루프가 있을 때마다 배열의 length 속성에 반복 액세스합니다.
더 나은 코드는 루프 외부의 length 속성에 액세스하여 루프 실행 속도가 빨라집니다.

표현식 1은 코드 블록 실행 전에 실행됩니다(한 번).
루프가 시작되기 전에 변수를 설정합니다(let i = 0).
표현식 2는 코드 블록을 실행하기 위한 조건을 정의합니다.
루프 실행 조건을 정의합니다(i는 5보다 작아야 함).
표현식 3은 코드 블록이 실행된 후 (매번) 실행됩니다.
루프의 코드 블록이 실행될 때마다 값(i++)을 증가시킵니다.

```
for (let i = 0; i < 5; i++) {
  text += "The number is " + i + "<br>";
}

var i = 5;
for (var i = 0; i < 10; i++) {
  실행문
}
```
var를 사용하여 루프에서 선언된 변수가 루프 외부에서 변수를 다시 선언합니다.
```
// Here i is 10

let i = 5;
for (let i = 0; i < 10; i++) {
  // some code
}
// Here i is 5
```

루프에서 i 변수를 선언하는 데 let을 사용하면 i 변수는 루프 내에서만 볼 수 있습니다.

JavaScript For In

JavaScript for in 문은 개체의 속성을 반복합니다.
```
for (key in object) {
  // 실행할 코드 블록
}
<p id="demo"></p>

<script>
const person = {fname:"John", lname:"Doe", age:25};

let txt = "";
for (let x in person) {
  txt += person[x] + " ";
}
document.getElementById("demo").innerHTML = txt;
```

For In Over Arrays

JavaScript for in 문은 배열의 속성을 반복할 수도 있습니다.
```
for (variable in array) {
  code
```

```
}
const numbers = [45, 4, 9, 16, 25];

let txt = "";
for (let x in numbers) {
  txt += numbers[x] + "<br>";
}

document.getElementById("demo").innerHTML = txt;
```

JavaScript While Loop

루프는 지정된 조건이 참인 한 코드 블록을 실행할 수 있습니다.

```
while (조건) {
  // 실행할 코드 블록
}
<script>
let text = "";
let i = 0;
while (i < 10) {
  text += "<br>The number is " + i;
  i++;
}
document.wrie(text);
</script>
```

The Do While Loop

do while 루프는 while 루프의 변형입니다. 이 루프는 조건이 참인지 확인하기 전에 코드 블록을 한 번 실행한 다음 조건이 참인 동안 루프를 반복합니다.

do while 루프는 while 루프의 변형입니다. 이 루프는 조건이 참인지 확인하기 전에 코드 블록을 한 번 실행한 다음 조건이 참인 동안 루프를 반복합니다.

```
<script>
let text = ""
let i = 0;
do {
  text += "<br>The number is " + i;
  i++;
}
while (i < 10);
document.write(text);
```

JavaScript Break and Continue

break 문을 사용하여 루프에서 빠져나올 수도 있습니다.

```
<script>
let text = "";
for (let i = 0; i < 10; i++) {
  if (i === 3) { break; }
  text += "The number is " + i + "<br>";
}
document.write(text);
</script>
```

break 문은 루프 카운터(i)가 3일 때 루프를 종료합니다(루프를 "중단").

The Continue Statement

continue 문은 지정된 조건이 발생하는 경우 (루프에서) 하나의 반복을 중단하고 루프의 다음 반복을 계속합니다.

```
<script>
let text = "";
for (let i = 0; i < 10; i++) {
  if (i === 3) { continue; }
  text += "The number is " + i + "<br>";
}
document.write(text);
</script>
```

JavaScript Functions

사용자 정의함수

JavaScript 함수는 특정 작업을 수행하도록 설계된 코드 블록입니다.
JavaScript 함수는 "무언가"가 함수를 호출(호출)할 때 실행됩니다.

JavaScript Function Syntax

```
function name(parameter1, parameter2, parameter3) {
  // code to be executed
}
```

세미콜론은 실행 가능한 JavaScript 문을 구분하는 데 사용됩니다.
함수선언은 실행 가능한 문이 아니기 때문에 세미콜론으로 끝내는 것은 일반적이지 않습니다.
JavaScript 함수는 function 키워드, 이름, 괄호()로 정의됩니다.
함수 이름에는 문자, 숫자, 밑줄 및 달러 기호(변수와 동일한 규칙)가 포함될 수 있습니다.

괄호는 쉼표로 구분된 매개변수 이름을 포함할 수 있습니다.
(매개변수1, 매개변수2, ...)
함수 호출에서 매개변수는 함수의 인수입니다.
JavaScript 인수는 값 으로 전달됩니다 .
함수는 인수의 위치가 아닌 값만 알게 됩니다.
함수가 인수의 값을 변경하더라도 매개변수의 원래 값은 변경되지 않습니다.
인수에 대한 변경 사항은 함수 외부에서 표시(반영)되지 않습니다.

함수에 의해 실행될 코드는 중괄호{} 안에 배치됩니다
함수 인수는 호출될 때 함수가 받는 값입니다.
함수 내에서 인수(매개 변수)는 지역 변수로 작동합니다.

함수 호출
1. 이벤트 발생 시(사용자가 버튼을 클릭할 때) 'onMouseOver',
2. JavaScript 코드에서 호출(호출)될 때
3. 자동으로(자체 호출)

Function Return - 함수 반환

JavaScript가 return 문에 도달하면 함수 실행이 중지됩니다.

함수가 문에서 호출된 경우 JavaScript는 호출문 다음에 코드를 실행하기 위해 "반환"합니다.

함수는 종종 반환 값을 계산합니다. 반환 값은 "호출자"에게 다시 "반환"됩니다.
함수 안에서 return을 만나게 되면 해당 함수를 호출한 곳으로 결과 데이터를 반환해 주고 함수는 종료가 된다.

함수 선언
선언된 함수는 즉시 실행되지 않습니다. 그것들은 "나중에 사용하기 위해 저장"되며 나중에 호출될 때 실행됩니다.

```
function myFunction(a, b) {
  return a * b;
}
```

함수 표현식
표현식 을 사용하여 JavaScript 함수를 정의할 수도 있습니다 .
함수 표현식은 변수에 저장할 수 있습니다.

```
const x = function (a, b) {return a * b};
```

함수 표현식이 변수에 저장된 후 변수를 함수로 사용할 수 있습니다.

```
const x = function (a, b) {return a * b};
let z = x(4, 3);
```
변수에 저장된 함수에는 함수 이름이 필요하지 않습니다. 항상 변수 이름을 사용하여 호출(호출)됩니다.

자체 호출 함수
함수 표현식은 "자기 호출"로 만들 수 있습니다.
자체 호출 식은 호출되지 않고 자동으로 호출(시작)됩니다.
표현식 뒤에 ()가 있으면 함수 표현식이 자동으로 실행됩니다.
함수 선언을 자체 호출할 수 없습니다.
함수식임을 나타내려면 함수 주위에 괄호를 추가해야 합니다.
```
<p id="demo"></p>

<script>
(function () {
  document.getElementById("demo").innerHTML = "Hello! I called myself";
})();
</script>
```
실제로 익명의 자체 호출 함수 (이름 없는 함수)입니다.

JavaScript 함수는 값으로 사용할 수 있습니다.
```
<p id="demo"></p>
<script>
function myFunction(a, b) {
  return a * b;
}
let x = myFunction(4, 3);
document.getElementById("demo").innerHTML = x;
</script>
```

함수는 객체입니다
arguments.length함수가 호출될 때 받은 인수 수를 반환합니다.
```
<p id="demo"></p>
<script>
function myFunction(a, b) {
  return arguments.length;
}
document.getElementById("demo").innerHTML = myFunction(4, 3);
</script>
```

함수를 사용하면 코드를 재사용할 수 있고, 코드를 한 번 정의하면 여러 번 사용할 수 있습니다. 다른 인수로 동일한 코드를 여러 번 사용하여 다른 결과를 생성할 수 있습니다.

```
<p id="demo"></p>
<script>
function myfnc(f) {
  return (5/9) * (f-32);
}
document.getElementById("demo").innerHTML = myfnc(77);
document.getElementById("demo").innerHTML = myfnc;
</script>
```

() 연산자가 함수를 호출합니다.
위의 예에서 myfnc는 함수 객체를 참조하고 myfnc()는 함수 결과를 참조합니다.

() 없이 함수에 액세스하면 함수 결과 대신 함수 개체가 반환됩니다.

전역변수와 지역변수
변수는 함수 블록{}을 기준으로 변수의 선언 위치에 따라 '전역 변수'와 '지역변수'로 나뉜다.
JavaScript 함수 내에서 선언된 변수는 함수에 대해 LOCAL이 됩니다.

지역변수
자바 스크립트 함수 내에서 선언 된 변수는 LOCAL 변수가 됩니다.
함수 내에서 지역 변수에만 액세스할 수 있습니다.
지역 변수는 함수 내에서만 인식되기 때문에 이름이 같은 변수를 다른 함수에서 사용할 수 있습니다.
지역 변수는 함수가 시작될 때 만들어지고 함수가 완료되면 삭제됩니다.

```
// code here can NOT use carName

function myFunction() {
  let carName = "Volvo";
  // code here CAN use carName
}
// code here can NOT use carName
```

전역 변수는 함수 블록{} 밖이나 안에서 자유롭게 사용 가능하지만 지역변수는 함수 블록{} 내에서만 사용할 수 있다.

전역변수	지역변수

let 변수;
function 함수() {
}
	function 함수() {
	let 변수;
	}

JavaScript Math Object

JavaScript Math 개체를 사용하면 숫자에 대한 수학적 작업을 수행할 수 있습니다.
다른 객체와 달리 Math 객체에는 생성자가 없습니다.
Math 객체를 먼저 생성하지 않고도 모든 메서드와 속성을 사용할 수 있습니다.

Math Properties

Math 속성의 구문은 Math.property입니다.

JavaScript는 Math 속성으로 액세스할 수 있는 8개의 수학 상수를 제공합니다.

ex)
Math.E // 오일러 수를 반환합니다.
Math.PI // PI 반환
Math.SQRT2 // 2의 제곱근을 반환합니다
Math.SQRT1_2 // 1/2의 제곱근을 반환합니다.
Math.LN2 // 2의 자연 로그를 반환합니다.
Math.LN10 // 10의 자연 로그를 반환합니다.
Math.LOG2E // E의 밑이 2인 로그를 반환합니다.
Math.LOG10E // E의 밑이 10인 로그를 반환합니다.

Math Methods

Math 모든 메서드의 구문은 Math.method(숫자)입니다.
숫자를 정수로 반올림하는 4가지 일반적인 방법이 있습니다.

Math.round(x)	가장 가까운 정수를 반환합니다.
Math.ceil(x)	가장 가까운 정수로 반올림한 x 값을 반환합니다.
Math.floor(x)	Math.floor(x)는 가장 가까운 정수로 내림한 x 값을 반환합니다.
Math.trunc(x)	x의 정수 부분을 반환합니다.

Math.sign()

x가 음수, null 또는 양수이면 반환합니다.

ex)

```
Math.sign(-4);
Math.sign(0);
Math.sign(4);
```

Math.pow()

Math.pow(x, y)는 x의 값을 y의 거듭제곱으로 반환합니다.

ex)

```
Math.pow(8, 2);
```

Math.sqrt()

Math.sqrt(x)는 x의 제곱근을 반환합니다.

ex)

```
Math.sqrt(64);
```

Math.abs()

Math.abs(x)는 x의 절대(양수) 값을 반환합니다.

ex)

```
Math.abs(-4.7);
```

Math.min() and Math.max()

Math.min() 및 Math.max()는 인수 목록에서 가장 낮거나 가장 높은 값을 찾는 데 사용할 수 있습니다.

ex)

```
Math.min(0, 150, 30, 20, -8, -200);
Math.max(0, 150, 30, 20, -8, -200);
```

Math.random()

Math.random()은 0(포함)과 1(제외) 사이의 난수를 반환합니다.

ex)

```
Math.random();
```

JavaScript Events

event	Description
onchange	HTML 요소가 변경되었습니다.
onclick	사용자가 HTML 요소를 클릭합니다.
onmouseover	사용자가 HTML 요소 위로 마우스를 이동합니다.
onmouseout	사용자가 HTML 요소에서 멀리 마우스를 이동합니다.
onkeydown	사용자가 키보드 키를 누릅니다
onload	브라우저가 페이지 로드를 완료했습니다.
onscroll	요소의 스크롤 막대가 스크롤될 때 발생합니다.
onmousemove	포인터가 요소 위에 있는 동안 포인터가 움직일 때 발생합니다.

이벤트 핸들러는 사용자 입력, 사용자 작업 및 브라우저 작업 등을 처리하고 확인하는 데 사용할 수 있습니다.

페이지가 로드 될 때마다해야 할 일
페이지를 닫을 때 수행해야 할 작업
사용자가 단추를 클릭할 때 수행해야 하는 작업
사용자가 데이터를 입력할 때 확인해야 하는 콘텐츠

JavaScript가 이벤트와 함께 작동하도록 하기 위해 다양한 방법을 사용할 수 있습니다.

HTML 이벤트 속성은 자바 스크립트 코드를 직접 실행할 수 있습니다.
HTML 이벤트 속성은 자바 스크립트 함수를 호출 할 수 있습니다.
사용자 고유의 이벤트 처리기 함수를 HTML 요소에 할당할 수 있습니다.
이벤트가 전송되거나 처리되지 않도록 할 수 있습니다.

onscroll Event

onscroll 이벤트는 요소의 스크롤 막대가 스크롤될 때 발생합니다.

Syntax
In HTML:
<element onscroll="myScript">
In JavaScript:
object.onscroll = function(){myScript};

scrollTop 속성은 요소의 콘텐츠가 세로로 스크롤되는 픽셀 수를 설정하거나 반환합니다.
scrollLeft 속성은 요소의 콘텐츠가 가로로 스크롤되는 픽셀 수를 설정하거나 반환합니다.
Syntax)
element.scrollTop
element.scrollLeft

ex)

```
<div id="myDIV" onscroll="myfnc()">
  <div id="content">Scroll me!</div>
</div>
<p id="demo"></p>
<script>
function myfnc() {
  const element = document.getElementById("myDIV");
  let x = element.scrollLeft;
```

```
    let y = element.scrollTop;
    document.getElementById ("demo").innerHTML = "Horizontally: " + x.toFixed() +
"<br>Vertically: " + y.toFixed();
}
</script>
```

ex)

```
<script>
        window.onscroll = function () {
          let ht = document.documentElement.scrollTop;
          console.log(ht);
          if (document.documentElement.scrollTop > 20) {
            document.getElementById("navbar").style.top = "0";
          } else {
            document.getElementById("navbar").style.top = "-50px";
          }
        };
</script>
```

onmousemove Event

onmousemove 이벤트는 포인터가 요소 위에 있는 동안 포인터가 움직일 때 발생합니다.
In HTML:
<element onmousemove="myScript">
Try it Yourself »
In JavaScript:
object.onmousemove = function(){myScript};

clientX, clientY속성은 브라우저가 기준인 좌표입니다. 현재 보이는 브라우저 화면 상에서 어느 지점에 위치하는 지를 의미하기 때문에 스크롤 해도 값은 변하지 않습니다.
clientX : 브라우저 페이지에서의 X좌표 위치를 반환합니다.
clientY : 브라우저 페이지에서의 Y좌표 위치를 반환합니다.

pageX, pageY속성은 문서가 기준입니다. client와 비슷하지만 이 메서드는 문서 전체 크기가 기준이라 스크롤 시 값이 바뀝니다. (스크롤을 포함해서 측정합니다)
pageX : 브라우저 페이지에서의 x좌표 위치를 반환합니다.
pageY : 브라우저 페이지에서의 Y좌표 위치를 반환합니다.

screenX, screenY속성은 화면 출력 크기(자신의 모니터)가 기준인 절대 좌표입니다. 브라우저를 움직여도 값은 같습니다.
screenX : 전체 모니터 스크린에서의 x좌표 위치를 반환합니다.
screenY : 전체 모니터 스크린에서의 y좌표 위치를 반환합니다.

The onload and onunload Events

onload 및 onunload 이벤트는 사용자가 페이지에 들어가거나 나갈 때 트리거 됩니다.

onload 이벤트는 방문자의 브라우저 유형 및 브라우저 버전을 확인하고 정보를 기반으로 웹 페이지의 적절한 버전을 로드하는 데 사용할 수 있습니다.

ex)

```
<script>
function mymessage() {
  alert("This message was triggered from the onload event");
}
</script>
</head>

<body onload="mymessage()">
</body>
```

The onchange Event

onchange 이벤트는 종종 입력 필드의 유효성 검사와 함께 사용됩니다.
다음은 사용자가 입력 필드의 내용을 변경할 때 upperCase() 함수가 호출됩니다.

ex)

```
<input type="text" id="fname" onchange="upperCase()">
<script>
function upperCase() {
  const x = document.getElementById("fname");
  x.value = x.value.toUpperCase();
}
</script>
```

Window Size

두 가지 속성을 사용하여 브라우저 창의 크기를 결정할 수 있습니다.

두 속성 모두 크기를 픽셀 단위로 반환합니다.
window.innerHeight - 브라우저 창의 내부 높이(픽셀 단위)
window.innerWidth - 브라우저 창의 내부 너비(픽셀 단위)

ex)

```
<p id="demo"></p>
<script>
```

```
document.getElementById("demo").innerHTML =
"Browser inner window width: " + window.innerWidth + "px<br>" +
"Browser inner window height: " + window.innerHeight + "px";
</script>

Set Text Content
<p id="demo" onclick="myfnc()">demo</p>
<script>
function myfnc() {
  document.getElementById("demo").textContent = "I have changed!";
}
</script>
```

preventDefault() Event Method

preventDefault() 메서드는 이벤트가 취소 가능한 경우 이벤트를 취소합니다.
즉, 이벤트에 속한 기본 작업이 발생하지 않습니다.
"제출" 버튼을 클릭하면 양식이 제출되지 않습니다.
링크를 클릭하면 링크가 URL을 따라가지 않도록 방지합니다.

JavaScript Errors

The try statement 실행할 코드 블록을 정의합니다.
The catch statement 오류를 처리하는 코드 블록을 정의합니다.
The finally statement 결과에 관계없이 실행할 코드 블록을 정의합니다.
The throw statement 사용자 지정 오류를 정의합니다.

```
<script>
function myFunction() {
  const message = document.getElementById("p01");
  message.innerHTML = "";
  let x = document.getElementById("demo").value;
  try {
    if(x == "")  throw "is empty";
    if(isNaN(x)) throw "is not a number";
    x = Number(x);
    if(x > 10)   throw "is too high";
    if(x < 5)  throw "is too low";
  }
  catch(err) {
    message.innerHTML = "Input " + err;
  }
```

```
  finally {
    document.getElementById("demo").value = "";
  }
}
</script>

<p>Please input a number between 5 and 10:</p>

<input id="demo" type="text">
<button type="button" onclick="myFunction()">Test Input</button>
<p id="p01"></p>
```