

# JavaScript

JavaScript 개요

## JavaScript 소개

---

- JavaScript는 굉장히 가벼운 프로그래밍 언어이다.
- 웹 애플리케이션 개발 시 많이 사용하며 클라이언트 프로그래밍을 위해 사용한다.
- 객체 지향 프로그래밍 언어이다.
- 국제 표준 규격을 가지고 있다.

## JavaScript 소개

---

- 1995년 Netscape 2.0에서 LiveScript라는 이름으로 발표되었다.
- 이후 웹 개발 분야에서 Java 언어가 크게 성공하자 이름을 JavaScript로 변경하였다.
- 현재 모든 웹 브라우저에서 채택하고 지원하고 있다.

## 사용자 측면에서의 JavaScript

---

- 브라우저가 해석하고 실행하는 코드가 HTML 문서에 포함된다.
- 웹 페이지가 정적인 경우에는 필요하지 않으나 사용자와 상호 작용을 하고 변화가 이루어질 경우 JavaScript로 작성한다.
- 웹 브라우저에서 발생하는 각종 이벤트나 입력 데이터에 대한 처리 등 클라이언트에서 이루어지는 다양한 작업들을 처리 할 수 있다.

## 서버 측면에서의 JavaScript

---

- JavaScript는 웹 브라우저에서 동작하는 클라이언트 측 프로그래밍 언어로 사용되어 왔다.
- 최근에는 다양한 분야에서 JavaScript 언어를 활용하려고 하고 있다.
- Node.js 같은 프로그래밍 분야를 활용하면 JavaScript를 통해 서버 애플리케이션을 개발 할 수 있다.

# JavaScript

기본 문법

## 세미콜론

---

- JavaScript는 문장의 마지막에 세미콜론(;)을 찍어 라인이 끝났다는걸 명시한다.
- 하지만 세미콜론은 필수는 아니다.
- 여러 라인의 코드를 한 번에 작성할 경우에는 세미 콜론을 반드시 붙여줘야 한다.

```
<script>
  document.write("세미콜론<br/>");
  document.write("세미콜론<br/>")

  document.write("세미콜론<br/>"); document.write("세미콜론<br/>")
</script>
```

## 대소문자

---

- JavaScript는 대소문자를 엄격하게 구분한다.
- 소문자로 만들어진 요소를 대문자로 작성하면 오류가 발생한다.

```
<script>  
    document.write("문자열<br/>");  
    DOCUMENT.WRITE("문자열<br/>");  
</script>
```



## 주석

---

- JavaScript 는 두 가지 주석을 제공한다
- `//` : 한 줄 주석
- `/* */` : 여러 줄 주석

```
<script>  
    // 이 부분은 한줄 주석입니다.  
    /*  
        이 부분은  
        여러줄  
        주석입니다  
    */  
</script>
```

# JavaScript

작성 위치

## inline 방식

---

- HTML 태그 내에 JavaScript 코드를 작성한다.
- 보통 태그에 대한 이벤트를 처리하기 위해 사용한다.

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

## internal 방식

---

- HTML 문서 내부에 JavaScript 코드를 작성하며 script 태그 사이에 작성한다.
- 함수를 정의하여 사용하거나 페이지가 나타날때 자동으로 실행되는 코드가 필요할 때 사용하면 된다.
- 코드는 위에서 아래 방향으로 동작한다.

```
<script>  
    JavaScript Code Here....  
</script>
```

## external 방식

---

- JavaScript 파일을 js 파일에 작성하고 이를 HTML 문서에 삽입하는 방식이다.
- 여러 파일에서 공통적으로 사용하는 코드가 있을 때 사용한다.

```
<script src="Test.js"></script>
```

# JavaScript

출력문

## JavaScript의 출력문

---

- innerHTML
- document.write() 함수
- alert () 함수
- console.log() 함수

## alert 함수

---

- 웹 브라우저에 경고창을 띄우는 함수

```
<script>  
  alert("메시지 출력");  
</script>
```

localhost:8080 내용:

메시지 출력

확인

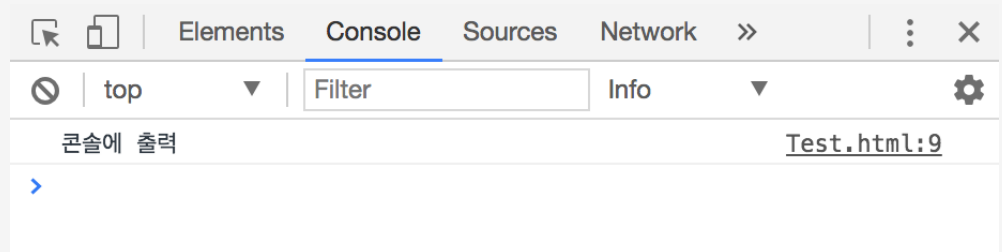


## console.log 함수 사용

---

- 웹 브라우저에서 제공하는 콘솔에 출력하는 함수

```
<script>  
    console.log("콘솔에 출력");  
</script>
```



## document.write 함수

---

- 지정한 문자열을 HTML 코드로 출력하는 함수

```
<script>  
    document.write("HTML 코드 출력");  
</script>
```

## innerHTML

---

- div 와 같이 내부에 HTML 코드를 넣을 수 있는 태그에 HTML 코드를 삽입하는 방식

```
<div id="test"></div>
<script>
    document.getElementById("test").innerHTML = "HTML코드 삽입";
</script>
```

# JavaScript

변수와 자료형

## 자료형

---

- 자바 스크립트는 다음과 같은 자료형을 제공한다.
- 정수 : 100
- 실수 : 10.1
- 문자열 : "문자열", '문자열'
- 참 거짓 : true, false

## 변수

---

- 변수는 데이터를 담는 기억장소를 의미한다.
- 자바 스크립트는 변수 선언시 자료형을 명시하지 않고 var라는 키워드를 사용한다.
- 변수에 담는 값의 자료형에 따라 값을 관리하는 방법을 자동으로 선택한다.

```
<script>
  var a1 = 100;
  var a2 = 10.1;
  var a3 = "문자열";
  var a4 = true;

  document.write("a1 : " + a1 + "<br/>");
  document.write("a2 : " + a2 + "<br/>");
  document.write("a3 : " + a3 + "<br/>");
  document.write("a4 : " + a4 + "<br/>");
</script>
```

undefined

---

- 변수를 선언하고 값을 대입하지 않으면 undefined 값이 자동으로 대입된다.

```
<script>  
    var a1;  
    document.write("a1 : " + a1 + "<br/>");  
</script>
```

a1 : undefined

## 캐스팅

---

- 데이터 형식을 변경한다.
- 강제 형변환 : Number(), Boolean(), String()
- 자동 형변환

숫자+문자 -> 문자

숫자(\*/%)문자 -> 숫자

```
var inData = prompt('숫자를 입력하세요...');  
console.log('inData = ', inData);  
console.log('inData 자료형은? ', typeof(inData));  
inData = Number(inData);  
console.log('inData = ', inData);  
console.log('inData 자료형은? ', typeof(inData));
```



# JavaScript

입력문과 선택상자

## 입력문 prompt()

---

- var 저장변수 = prompt(메세지,초기값);
- 결과값은 문자열 string

```
var userName = prompt("고객님의 이름을 입력하세요");  
alert(userName+"님 환영합니다.");
```

## 선택상자 confirm()

---

- 결과값은 boolean (true/false)
- var 저장변수 = confirm(메세지)

```
var ans = confirm("자바스크립트는 인터프리터 언어이다");  
document.write(ans? "정답":"오답");
```

# JavaScript

연산자

## 자료형

---

- 자바 스크립트는 다음과 같은 연산자를 제공한다.
- 산술연산자
- 대입연산자
- 비교연산자
- 논리연산자
- 타입연산자
- 비트연산자

# 산술연산자

---

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
++	Increment
--	Decrement

## 대입연산자

---

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

## 비교연산자

---

Operator	Description
==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
?	ternary operator



## 논리연산자

---

Operator	Description
&&	logical and
	logical or
!	logical not

## 타입연산자

---

Operator	Description
typeof	Returns the type of a variable
instanceof	Returns true if an object is an instance of an object type

## 연산자 실습

---

```
// +, -, *, /, %(나머지연산자), **
var x=70, y=6;
console.log(x, '+', y, '=', x+y);
console.log(x, '*', y, '=', x*y);
console.log(x, '/', y, '=', x/y);
console.log(x, '%', y, '=', x%y);
console.log(x, '**', y, '=', x**y);
```

```
// 증감(전위, 후위 연산자) : 숫자형변수에 사용
// ++변수, 변수++, --변수, 변수-- : 값이 1씩 증가하거나 1씩 감소
var z=0;
console.log('z = ', z); // z = 0
console.log('++z ', ++z); // ++z 1
console.log('z++ ', z++); // z++ 1
console.log('z = ', z); // z = 2
console.log('--z ', --z); // --z 1
console.log('z-- ', z--); // z-- 1
console.log('z = ', z); // z = 0
```

## 연산자 실습

---

```
// 대입연산자 : 숫자형에서 사용
// +=, -=, *=, /=
var numA = 10;
var numB = 20;
// numA=numA+numB;
numA+=numB;
console.log('numA = ',numA); // numA = 30
// numA=numA-numB;
numA-=numB;
console.log('numA = ',numA); // numA = 10

// 문자열 연산자 + (연결 연산자)
var message1 = '메리', message2='크리스마스';
console.log('message1+message2 = ',message1+message2);
```

## 연산자 실습

---

```
// 관계(비교)연산자 => true/false
// 변수1(>/</>=/<=)변수2,
// ==(데이터형은 상관없이 값만 같다), !=
// ===(데이터형도 같고 값도 같다.), !==
var num3=100, num4=30, num5='30';
console.log('num3>num4', num3>num4); // num3>num4 true
console.log('num3==num4', num3==num4); // num3==num4 false
console.log('num3!=num4', num3!=num4); // num3!=num4 true
console.log('num4==num5', num4==num5);
// num4==num5 true
console.log('num4===num5', num4===num5);
// num4===num5 false
```

## 연산자 실습

---

```
// 논리연산자 => true/false
// || : 논리의합 or , 조건 중 하나만 true이면 true
// && : 논리의곱 and , 조건이 모두 true이어야 true
// ! : 부정 not
console.log('true || false = ', true || false);
console.log('true || true = ', true || true);
console.log('false || false = ', false || false);
console.log('true && false = ', true && false);
console.log('true && true = ', true && true);
console.log('false && false = ', false && false);
console.log('!false = ', !false );
```

```
var i=100, j=300, k=100;
console.log('(i>j)||(i==k) = ', (i>j)||(i==k));
console.log('(i!=j)||(i!=k) = ', (i!=j)||(i!=k));
console.log('(i>j)&&(i==k) = ', (i>j)&&(i==k));
console.log('(i<j)&&(i==k) = ', (i<j)&&(i==k));
console.log('!(i<j) = ', !(i<j));
```

## 연산자 실습

---

```
// 조건 연산자 = 삼항 연산자
// 조건if문과 비슷
// 조건? 명령1:명령2
// 조건이 true 이면 명령1
// 조건이 false 이면 명령2
// true? alert('참'):alert('거짓'); // alert('참')
// false? alert('참'):alert('거짓'); // alert('거짓')

var l=50, m=50, n=300;
(m>n)? console.log('m이 크다'):console.log('n이 크다');
// n이 크다
((l==m) && (m>n)) ? console.log('참'):console.log('거짓');
// 거짓
((l==m) || (m>n)) ? console.log('참'):console.log('거짓');
// 참
```

# JavaScript

분기문



## 제어문

---

- 코드는 위에서 아래로 실행되는데 이러한 흐름을 변경할 때 사용하는 것을 제어문이라고 한다.
- 제어문은 두 가지로 구분된다.
- 분기문 : if, switch
- 반복문 : for, while, do while

## if 문

---

- if 문은 분기문 중에 가장 많이 사용하는 문장이다.
- 주어진 조건에 만족하면 실행되고 만족하지 않으면 실행되지 않는다.

```
var a = 10;  
  
if(a > 20){  
    document.write("a는 20보다 큽니다");  
}  
  
if(a < 20){  
    document.write("a는 20보다 작습니다");  
}
```

## else 문

---

- if 문의 조건이 만족하지 않을 경우 실행될 부분이다.

```
var a = 10;

if(a > 20){
    document.write("a는 20보다 큽니다");
} else {
    document.write("a는 20크지 않습니다.");
}
```

## else if 문

---

- 다중 조건을 가졌을 때 사용한다. 조건 중 참에 해당하는 부분만 실행되며 위에서부터 조건 검사를 하게 된다.
- 만약 모든 조건에 만족하지 않을 경우도 필요하다면 마지막에 else 문을 넣어준다.

```
var a = 10;

if(a > 20){
    document.write("a는 20보다 큼니다");
} else if(a > 10){
    document.write("a는 10보다 큼니다");
} else if(a > 0){
    document.write("a는 0보다 큼니다");
} else {
    document.write("a는 0보다 크지 않습니다");
}
```

## switch 문

---

- 변수나 수식에 해당하는 값 부분으로 이동하여 코드가 실행되는 분기 문 이다.

```
switch (수식/변수) {  
    case 값1:  
        명령문1  
        break  
    case 값2:  
        명령문2  
        break  
    case 값3:  
        명령문3  
        break  
    default:  
        명령문4  
}  
명령문5
```

## switch 문

---

- 모든 조건을 검사하면서 내려오는 if문과 달리 바로 이동하여 실행하므로 if문 보다 빠르다.

```
var a = 10;

if(a == 20){
    document.write("a는 20입니다");
} else if(a == 10){
    document.write("a는 10입니");
} else if(a == 0){
    document.write("a는 0입니다");
} else {
    document.write("a는 20, 10, 0이 아닙니다");
}
```

```
switch(a){
case 20 :
    document.write("a는 20입니다");
    break;
case 10 :
    document.write("a는 10입니다");
    break;
case 0 :
    document.write("a는 0입니다");
    break;
default :
    document.write("a는 20, 10, 0이 아닙니다");
}
```

## switch 문

---

```
// switch case 문을 이용한 짝수, 홀수, 에러
var n = 5;
switch (n % 2) {
  case 0:
    console.log('짝수');
    break;
  case 1:
    console.log('홀수');
    break;
  default:
    console.log('에러')
}
console.log('테스트1 끝')
```

```
// break 문이 없는 switch ... case 테스트
// break 문이 없다면 모든 명령문이 실행된다.
var ans = 'yes'
switch (ans) {
  case ('yes'):
    console.log('yes')
  case ('no'):
    console.log('no')
  default:
    console.log('default')
}
console.log('테스트2 끝')
```

# JavaScript

반복문



## while 문

---

- 주어진 조건에 만족할 경우 코드를 반복하는 반복문 인다.
- 반복할 조건이 결정되어 있을 때 사용한다.
- 조건이 처음부터 거짓이면 단 한번도 수행되지 않는다.

```
while(조건식) {  
    | | 명령문1  
}  
명령문2
```

```
while(a < 10){  
    a++;  
    document.write("while문 실행<br/>");  
}
```

## continue 문

---

- continue 문을 삽입하면 삽입된 위치 아래의 명령은 실행되지 않는다.

```
// 1~25까지에서 5의 배수만 출력하지 않는다.  
var cnt7 = 0  
document.write('<p>')  
while (cnt7 < 25) {  
    cnt7++;  
    if (cnt7 % 5 == 0) continue;  
    document.write(cnt7 + '<br />')  
}  
document.write('</p>')
```

## do while 문

---

- 주어진 조건에 만족할 경우 코드를 반복하는 반복문 인다.
- 반복할 조건이 결정되어 있을 때 사용한다.
- 조건이 처음부터 거짓이면 한번 수행된다.

```
초기치
do {
    명령문1
    증감
} while(조건식);
명령문2
```

```
do{
    a++;
    document.write("while문 실행<br/>");
} while(a < 10);
```

## do while 문

---

- while 문과 do~while 문 비교

```
while (false) {  
  document.write('명령실행1');  
}  
document.write('<p>while 테스트 종료</p>')
```

```
do {  
  document.write('명령실행2');  
} while (false);  
document.write('<p>do while 테스트 종료</p>')
```

## for 문

---

- 주어진 조건에 만족할 경우 코드를 반복하는 반복문이다.
- 주로 반복 횟수가 결정되어 있을 때 사용한다.

```
for (초기치;조건;증감) {  
  | 명령문  
}
```

```
for(var i = 0 ; i < 10 ; i++){  
  document.write("for문의 코드 입니다<br/>");  
}
```

## for 문

---

```
// 1~10까지 출력
for (var i = 1; i <= 10; i++) {
  console.log(i);
}
console.log('for 테스트1')
```

```
// 10~1까지 출력
for (var i = 10; i > 0; i--) {
  console.log(i);
}
console.log('for 테스트2')
```

## for 문

---

```
// for문안에 조건문 사용하기
// 10의 배수만 표시 (1~100)
for (var i = 1; i <= 100; i++) {
    if (i % 10 == 0) {
        document.write(i + ' ');
    }
}
```

```
// for문안에 continue문 사용하기
// 5의 배수와 7의 배수는 출력제외. 나머지는 출력(1~100)
document.write('<p>');
for (var i = 1; i <= 100; i++) {
    if ((i % 5 == 0) || (i % 7 == 0)) continue;
    document.write(i + ' ');
}
document.write('</p>');
```

## for 문

---

```
// for 문안에 for 문 = 중첩 for 문
document.write('<p>');
for (var i = 1; i <= 10; i++) {
    document.write(i + '<br />');
    for (var j = 1; j <= 5; j++) {
        document.write(j + ' ');
    }
    document.write('<br />')
}
document.write('</p>');
```



## for 문

---

```
* * *
* * *
* * *
* * *
* * *
* * *
* * *
```

```
document.write('<div>')
for (i = 1; i <= 7; i++) {
    for (j = 1; j <= 3; j++) {
        document.write(' * ');
    }
    document.write('<br />')
}
document.write('</div>')
```

## for 문

---

```
* * *
* * *
* * *
* * *
* * *
* * *
* * *
```

```
document.write('<div>')
for (i = 1; i <= 7; i++) {
    for (j = 1; j <= 3; j++) {
        document.write(' * ');
    }
    document.write('<br />')
}
document.write('</div>')
```

## for 문

---

// 중첩 for 문을 이용해서 테이블 만들기 1

```
document.write('<table style="margin-top:20px" border="1" cellpadding="10">')
for (var i = 1; i <= 3; i++) {
  document.write('<tr>');
  for (var j = 1; j <= 3; j++) {
    document.write('<td>' + i + ' - ' + j + '</td>');
  }
  document.write('</tr>');
}
document.write('</table>');
```

1 - 1	1 - 2	1 - 3
2 - 1	2 - 2	2 - 3
3 - 1	3 - 2	3 - 3

# JavaScript

배열

## 배열

---

- 여러 기억공간을 하나의 이름으로 관리할 때 사용한다.
- JavaScript는 [ ] 로 배열을 표현한다.

```
var array = [10, 20, 30, 40, 50];
```

## 배열의 개수

---

- 배열이 관리하는 기억장소의 개수는 length 라는 것을 이용해 파악할 수 있다.

```
var array = [10, 20, 30, 40, 50];
```

```
document.write("array의 개수 : " + array.length + "<br/>");
```

## 배열 요소 접근

---

- 배열이름[인덱스] 형태로 작성하면 배열 요소에 접근할 수 있다.
- 인덱스는 0 부터 시작한다.

```
var array = [10, 20, 30, 40, 50];

document.write("array[0] : " + array[0] + "<br/>");
document.write("array[1] : " + array[1] + "<br/>");

array[2] = 300;
array[3] = 400;

document.write("array[2] : " + array[2] + "<br/>");
document.write("array[3] : " + array[3] + "<br/>");
```

## 배열 생성

---

- []를 이용해서 빈배열을 만들고 아이템값 삽입

var 배열변수 = [];

배열변수[인덱스] = 값;

```
var arr1 = [];  
  
// 배열 전체 출력  
console.log('arr1 = ', arr1);  
  
// 배열값 추가  
arr1[0] = 'JS';  
arr1[1] = 100;  
console.log('arr1 = ', arr1);  
arr1[2] = true;  
console.log('arr1 = ', arr1);
```



## 배열 생성

---

- 배열을 생성할때 아이템값을 함께 지정
- var 배열변수 = [값1, 값2, 값3...];

```
var arr2 = [100, 200, 300, 400];  
console.log('arr2 = ', arr2);  
console.log('arr2의 전체길이는? ', arr2.length);  
console.log('arr2의 첫번째 아이템값? ', arr2[0]);  
console.log('arr2의 마지막 아이템값? ', arr2[arr2.length - 1]);
```

## 배열 생성

---

- new Array()를 이용해서 빈배열을 만들고 아이템값 삽입
- var 배열변수 = new Array();
- 배열변수[인덱스] = 값;

```
var arr3 = new Array();  
console.log('arr3 = ', arr3);  
arr3[0] = '신데렐라';  
arr3[arr3.length] = '백설공주';  
arr3[arr3.length] = '장화홍련전';  
console.log('arr3 = ', arr3);  
console.log('arr3의 전체길이는? ', arr3.length);
```

## 배열 생성

---

- new Array() 로 배열생성시 초기 아이템값 함께 삽입
- var 배열변수 = new Array(값1, 값2, 값3...);

```
var arr4 = new Array('트와이스', '방탄소년단');  
console.log('arr4 = ', arr4);  
arr4[arr4.length] = '엑소';  
arr4[arr4.length] = '레드벨벳';  
console.log('arr4 = ', arr4);
```

## 배열 생성

---

```
// 배열 중에서 홀수번째만 출력하기 (for, if 문 이용)
var arr2 = ['이영희', '김철수', '소나영', '김민주', '최민수',
            '홍길동', '송은주', '신숙자'];
document.write('<p>arr2 = ' + arr2 + '</p>');
document.write('<p>');
for (var i = 0; i < arr2.length; i++) {
    document.write(arr2[i] + '<br />');
}
document.write('</p>');

document.write('<p> 홀수번째만 출력 <br />');
for (var i = 0; i < arr2.length; i++) {
    if (i % 2 == 0) {
        document.write((i + 1) + '번째 : ' + arr2[i] + '<br />');
    }
}
document.write('</p>');
```

## for in

---

- 배열이 관리하는 기억장소를 처음부터 끝까지 순회할 경우 for 문을 사용한다.
- for in은 배열 순회를 목적으로 for문을 사용할 때 보다 편하게 사용할 수 있도록 지원하는 개념이다.

```
var array = [10, 20, 30, 40, 50];

for(var i = 0 ; i < array.length ; i++){
    document.write("array : " + array[i] + "<br/>");
}

for(var idx in array){
    document.write("array : " + array[idx] + "<br/>");
}
```

## 배열 메서드

---

### 데이터 변경

- push(): 배열의 끝에 값을 추가.
- pop(): 배열 마지막 값을 제거.
- shift(): 배열 데이터를 왼쪽으로 하나씩 밀어 맨앞 값을 제거.
- splice(): 배열값을 추가하거나 제거해서 반환.
- reverse(): 배열을 역순으로 재배치.
- sort(): 배열 데이터를 정렬.

## 배열 메서드

---

### 배열의 일부를 반환

`push()`: 배열의 끝에 값을 추가.

- `concat()`: 두개의 배열을 합침.
- `join()`: 배열 데이터 사이에 원하는 문자열을 넣어 구분자로 사용.
- `slice()`: 배열의 일부를 지정해서 가져옴.

## 배열 메서드

---

### 배열의 정렬

sort() , reverse()

숫자 < 알파벳대문자 < 알파벳소문자 < 한글

```
var arr1 = [100, 'word', 500, 10, 'spring', '봄', 1, '기차', 'JS', 'js'];
document.write('<p> arr1 => ' + arr1 + '</p>');
arr1.sort();
document.write('<p> sort() 적용 => ' + arr1 + '</p>');
arr1.reverse();
document.write('<p> reverse() 적용 => ' + arr1 + '</p>');
```



## 배열 메서드

---

### 배열안에 특정값이 있는지 확인하기

`indexOf(값1, 값2)` : 위치값이 반환. 없으면 -1

`includes(값)` : true / false 결과값

# JavaScript

함수

## 함수

---

- 개발자가 필요할 때 동작시킬 수 있도록 만들어 놓는 코드블럭
- 브라우저는 함수의 존재만 파악하고 있다가 개발자가 원할 때 함수 내부에 작성해 놓은 코드를 실행한다.

```
function f1(){  
    document.write("함수가 호출되었습니다<br/>");  
}  
  
f1();
```

## 함수의 이름은 함수의 주소값

---

- 자바 스크립트는 함수의 이름이 함수가 존재하는 메모리의 주소 값을 담은 변수로 취급한다.
- 자바 스크립트에서는 함수의 주소 값을 다른 변수에 담는 것이 가능하다.

```
function f1(){  
    document.write("함수가 호출되었습니다<br/>");  
}  
  
var f2 = f1;  
f2();
```

## 익명함수

---

- 이름이 없는 함수를 익명 함수라고 한다.
- 이름이 없는 함수 이므로 호출을 위해 변수에 담아야 한다.
- 매개 변수로 함수를 넘길 때 주로 사용한다.

```
var f3 = function(){  
    document.write("익명함수 호출<br/>");  
}  
  
f3();
```

## 매개 변수

---

- 함수를 호출할 때 값을 넘겨줄 수 있으며 이 값은 매개 변수로 받을 수 있다.
- 함수 호출 시 넘겨주는 값의 개수와 정의된 매개 변수의 개수와는 무관하다.

```
function f4(a, b){  
    document.write("f4호출 <br/>");  
    document.write("a : " + a + "<br/>");  
    document.write("b : " + b + "<br/>");  
}  
  
f4();  
f4(10);  
f4(10, 20);  
f4(10, 20, 30, 40, 50);
```

## arguments 배열

---

- 함수를 호출할 때 넘겨주는 값은 모두 매개 변수로 받을 수 있다.
- 그 외에 자바스크립트의 모든 함수들은 arguments라는 배열이 자동으로 만들어지는데 여기에도 호출 시 넘겨주는 값들이 들어 있다.

```
function f5(){
    for(var idx in arguments){
        document.write(arguments[idx] + "<br/>");
    }
}

f5();
f5(10);
f5(10, 20);
f5(10, 20, 30, 40, 50);
```

# JavaScript

String 객체



## String 객체

---

- 문자열을 관리할 수 있는 기능을 가지고 있는 객체
- 문자열에 대한 다양한 작업을 할 수 있다.
- JavaScript에서는 " " 나 ' ' 로 묶은 모든 문자열은 String 객체에 해당한다.

## String 객체

---

- length : 문자열의 글자 수
- indexOf : 주어진 문자열의 위치를 반환 (존재하지 않으면 -1)
- substring : 문자열 일부를 반환
- replace : 문자열을 변경
- toUpperCase : 소문자 -> 대문자
- toLowerCase : 대문자 -> 소문자
- concat : 문자열 합치기

## String 객체

---

- split : 구분자를 활용하여 문자열 나누기

# JavaScript

Math 객체

## Math 객체

---

- 수학에 관련된 다양한 기능을 제공하는 객체이다
- PI : 원주율 값
- round : 소수점 이하 반올림
- abs : 절대값
- ceil : 소수점 이하 올림
- floor : 소수점 이하 버림
- min : 최소값

## Math 객체

---

- max : 최대 값
- random : 랜덤

# JavaScript

Date 객체

## Date 객체

---

- 시간과 날짜를 관리할 수 있는 객체이다
- `getFullYear` : 년도를 가져온다.
- `getMonth` : 월을 가져온다.
- `getDate` : 일을 가져온다.
- `getHours` : 시를 가져온다.
- `getMinutes` : 분을 가져온다.
- `getSeconds` : 초를 가져온다.
- `getMilliseconds` : 밀리초를 가져온다.

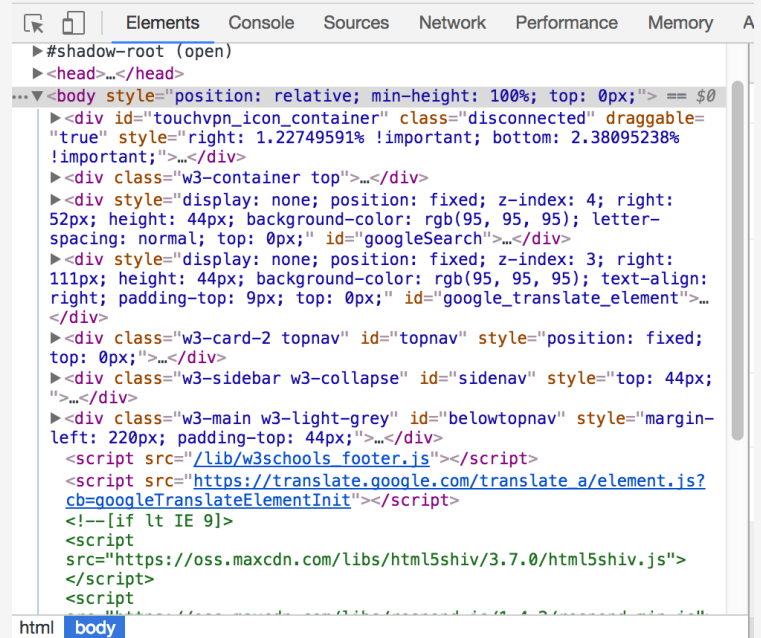


# JavaScript

DOM 1

# DOM

- Document Object Model 의 약자
- HTML 문서 구성을 위해 만들어지는 DOM 코드를 제어하고 관리하는 객체
- DOM을 이용해 DOM 코드를 제어하면 브라우저 화면에 반영시킬 수 있다.



The screenshot shows a web browser's developer tools interface. The 'Elements' tab is selected, displaying the DOM tree. The tree structure is as follows:

- #shadow-root (open)
  - <head>...</head>
  - <body style="position: relative; min-height: 100%; top: 0px;"> == \$0
    - <div id="touchvpn\_icon\_container" class="disconnected" draggable="true" style="right: 1.22749591% !important; bottom: 2.38095238% !important;">...</div>
    - <div class="w3-container top">...</div>
    - <div style="display: none; position: fixed; z-index: 4; right: 52px; height: 44px; background-color: rgb(95, 95, 95); letter-spacing: normal; top: 0px;" id="googleSearch">...</div>
    - <div style="display: none; position: fixed; z-index: 3; right: 111px; height: 44px; background-color: rgb(95, 95, 95); text-align: right; padding-top: 9px; top: 0px;" id="google\_translate\_element">...</div>
    - <div class="w3-card-2 topnav" id="topnav" style="position: fixed; top: 0px;">...</div>
    - <div class="w3-sidebar w3-collapse" id="sidenav" style="top: 44px;">...</div>
    - <div class="w3-main w3-light-grey" id="belowtopnav" style="margin-left: 220px; padding-top: 44px;">...</div>
    - <script src="/lib/w3schools\_footer.js"></script>
    - <script src="https://translate.google.com/translate\_a/element.js?cb=googleTranslateElementInit"></script>
    - <!--[if lt IE 9]>
    - <script src="https://oss.maxcdn.com/libs/html5shiv/3.7.0/html5shiv.js"></script>
    - <script>

The bottom of the interface shows the 'html' document type and the 'body' element selected.

## Document 객체

---

- innerHTML : 태그 내부에 html 코드를 설정한다.
- getElementsByTagName : 태그 이름을 통해 태그 객체를 가져온다
- getElementById : id 속성을 통해 태그 객체를 가져온다.
- getElementsByClassName : class 속성을 통해 태그 객체를 가져온다.

## innerHTML

---

- innerHTML 속성은 태그 내의 html 코드를 의미한다.
- 이 속성을 이용해 html 코드를 설정할 수 있다.

```
var test1 = document.getElementById("test1");  
  
test1.innerHTML = "<a href='http://android.com'>"  
                  + "<img src='image/android.png' />"  
                  + "</a>";
```

## 태그 만들기

---

- createElement : 태그를 생성한다.
- createTextNode : 문자열 노드를 생성한다.
- appendChild : 태그 내부에 다른 태그를 추가한다.

```
var a1 = document.createElement("a");
a1.href = "http://android.com";
a1.style.color = "red";

var a2 = document.createTextNode("안드로이드");
a1.appendChild(a2);

var a3 = document.createElement("img");
a3.src = "image/android.png";
a1.appendChild(a3);

var test1 = document.getElementById("test1");
test1.appendChild(a1);
```

# JavaScript

이벤트

## 이벤트

---

- HTML 문서에서 브라우저에 의해 발생하거나 사용자에게 의해 발생하는 사건들을 이벤트라고 부른다.
- JavaScript에서는 여러 이벤트에 대응하여 개발자가 작성한 코드가 수행될 수 있도록 다양한 이벤트를 정의하고 있다.

## 이벤트의 종류

---

- JavaScript는 마우스, 키보드 등 다양한 이벤트를 제공하고 있다.
- 이벤트의 종류는 다음 사이트를 참조한다.
- [https://www.w3schools.com/jsref/dom\\_obj\\_event.asp](https://www.w3schools.com/jsref/dom_obj_event.asp)



# JavaScript

BOM 1

## BOM

---

- Browser Object Model 의 약자.
- 웹 브라우저에 관한 정보를 파악하거나 여러 처리를 할 수 있도록 만들어진 객체들

## Window

---

- 브라우저에서 눈에 보이는 화면 부분과 관련된 객체
- innerWidth : 화면의 가로 길이
- innerHeight : 화면의 세로 길이
- alert : 경고창을 띄운다.
- confirm : 확인/취소를 선택할 수 있는 창을 띄운다.
- prompt : 입력창을 띄운다.
- open : 팝업창을 띄운다.

## Location

---

- 브라우저의 주소창과 관련된 객체이다.
- href : 주소창의 주소를 관리한다.

## Navigator

---

- 브라우저의 정보를 관리하는 객체이다.
- `cookieEnabled` : 쿠키 사용 여부
- `appName` : 브라우저 애플리케이션 이름
- `appCodeName` : 브라우저 코드 네임
- `product` : 브라우저 엔진 네임
- `appVersion` : 브라우저 버전
- `userAgent` : 브라우저의 종합 정보

## window.onload

---

- HTML 문서에 있는 모든 태그들의 객체가 생성되면 load 사건이 발생한다.
- window.onload에 설정되어 있는 함수는 load 사건이 발생하면 자동으로 호출이 되며 여기에서는 html 문서의 모든 태그 객체를 사용하는 것이 가능하다.

```
window.onload = function(){  
    document.write("load 사건 발생<br/>");  
}
```