JavaScript

연산자 – 김근형 강사

연산자

- JavaScript 연산자
  - 산술 연산자
  - 할당 연산자
  - 비교 연산자
  - 비트 연산자
  - 논리 연산자
  - 문자열 연산자
  - 조건 (삼항) 연산자
  - 쉼표 연산자
  - 단항 연산자
  - 관계 연산자

산술연산자

- 산술연산자

연산자	이름	설명
+	덧셈	덧셈 연산자는 숫자 피연산자를 더한 값, 또는 문자열을 연결한 값을 생성한다.
-	뺄셈	뺄셈 연산자는 두 개의 피연산자를 뺀 값을 생성한다.
/	나눗셈	나눗셈 연산자는 왼쪽 피연산자를 피제수로, 오른쪽 피연산자를 제수로 한 몫을 생성한다.
*	곱셈	곱셈 연산자는 두 연산자의 곱을 생성한다.
%	나머지	나머지 연산자는 왼쪽 피연산자를 오른쪽 피연산자로 나눴을 때의 나머지를 반환한다. 결과는 항상 피제수의 부호를 따라간다.
**	거듭제곱	거듭제곱 연산자는 첫 번째 피연산자를 밑으로, 두 번째 피연산자를 지수로 한 값을 생성한다.

산술연산자

- 산술연산자
  - Boolean의 덧셈 연산은 true는 1, false는 0으로 숫자와 결합한다.
  - String의 덧셈 연산은 다른 값들은 전부 String으로 결합 연산 한다.
  - 문자열을 마이너스 연산할 경우 NaN을 출력한다.

```
// Number + Number -> 합
console.log(1 + 2); // 3

// Boolean + Number -> 합
console.log(true + 1); // 2

// Boolean + Boolean -> 합
console.log(false + false); // 0

// Number + String -> 연결
console.log(5 + "foo"); // "5foo"

// String + Boolean -> 연결
console.log("foo" + false); // "foofalse"

// String + String -> 연결
console.log("foo" + "bar"); // "foobar"
```

```
3
2
0
5foo
foofalse
foobar
```

```
console.log(5 - 3); // 2
console.log(3 - 5); // -2
console.log("foo" - 3) // NaN
```

```
2
-2
NaN
```

산술연산자

- 산술연산자
  - 곱셉 연산 시 Infinite 연산은 NaN이 나오며 Infinite 끼리 연산은 Infinite가 나온다.
  - 나눗셈에서 정수와 정수의 연산의 결과는 정수가 아닌 실수가 나온다.
  - 나눗셈에서 0으로 나눌 경우 Infinity가 나온다.

```
console.log(2 * 2); // 4
console.log(-2 * 2); // -4
console.log(Infinity * 0); // NaN
console.log(Infinity * Infinity); // Infinity
console.log("foo" * 2); // NaN
```

```
4
-4
NaN
Infinity
NaN
```

```
console.log(1 / 2);      // JavaScript에선 0.5
// (양쪽 피연산자 모두 명시적인 부동소수점 숫자가 아님)
console.log(1.0 / 2.0);  // JavaScript와 Java 둘 다 0.5
console.log(2.0 / 0);    // JavaScript에서 Infinity
console.log(2.0 / 0.0);  // 동일하게 Infinity
console.log(2.0 / -0.0); // JavaScript에서 -Infinity
```

```
0.5
0.5
Infinity
Infinity
-Infinity
```

산술연산자

- 산술연산자
  - 나머지의 결과는 항상 피제수의 부호를 따라간다.
  - 거듭제곱 연산자는 우결합성 을 가진다. 따라서 a ** b ** c는 a ** (b ** c)와 같다.

```
console.log(12 % 5); // 2
console.log(-1 % 2); // -1
console.log(NaN % 2); // NaN
console.log(1 % 2); // 1
console.log(2 % 3); // 2
console.log(-4 % 2); // -0
console.log(5.5 % 2); // 1.5
```

```
console.log(2 ** 3); // 8
console.log(3 ** 2); // 9
console.log(3 ** 2.5); // 15.588457268119896
console.log(10 ** -1); // 0.1
console.log(NaN ** 2); // NaN

console.log(2 ** 3 ** 2); // 512
console.log(2 ** (3 ** 2)); // 512
console.log((2 ** 3) ** 2); // 64
```

산술연산자

- 산술연산자

```
console.log(-2 ** 2);
// Bash에서 4, 다른 언어에서는 -4.
// 모호한 표현이므로 JavaScript에서는 유효하지 않음

console.log(-(2 ** 2));
// JavaScript에서 -4, 작성자의 의도가 명확함

console.log(-(2 ** 2)); // -4
// 결과의 부호를 뒤집으려면 다음과 같이 작성한다.

console.log((-2) ** 2); // 4
// 거듭제곱의 밑을 음수로 강제하려면 다음과 같이 작성한다.
```

산술연산자

- 단항 부정/ 단항 양부호
  - 단항 부정 연산자는 피연산자의 앞에 위치하며 부호를 뒤집는다.
  - 단항 양부호 연산자는 피연산자의 앞에 위치하며 피연산자의 값 그대로 평가되지만, 값이 숫자가 아닐 경우 숫자로 변환을 시도한다.
  - 단항 부정(-) 연산자도 숫자가 아닌 값을 변환할 수 있지만, 단항 양부호가 속도도 제일 빠르고 다른 연산도 수행하지 않으므로 선호해야 할 방법이다.
  - 문자열로 표현한 정수 및 부동소수점 실수, 문자열이 아닌 true, false, null도 변환할 수 있다. 10진수와 (앞에 "0x"가 붙은) 16진수 정수 모두 지원한다. 음수도 지원하지만 16진수 음수에는 사용할 수 없다. 어떤 값을 분석할 수 없으면 NaN으로 평가된다.

산술연산자

- 단항 부정/ 단항 양부호

```
var x = 3;
y = -x; // y = -3, x = 3
console.log(y);

var x = "4";
y = -x; // y = -4
console.log(y);
```

```
-3
-4
```

```
console.log(+3);      // 3
console.log((+"3")+1);   // 4
console.log(+true);  // 1
console.log(+false); // 0
console.log(+null);  // 0
```

산술연산자

- 증감연산자
  - 증감연산자는 1을 증가시키거나 1을 감소시킬 경우 사용한다.
  - 피연산자 뒤에 붙여(예: x++, x--) 접미사로 사용한 경우 증가하기 전의 값을 반환한다.
  - 피연산자 앞에 붙여(예: ++x, --x) 접두사로 사용한 경우 증가한 후의 값을 반환한다.

```
// 접미사
var x = 3;
y = x++; // y = 3, x = 4
console.log("x : "+x+"   y : "+y);

// 접두사
var a = 2;
b = ++a; // a = 3, b = 3
console.log("a : "+a+"   b : "+b);

// 접미사
var x = 3;
y = x--; // y = 3, x = 2
console.log("x : "+x+"   y : "+y);

// 접두사
var a = 2;
b = --a; // a = 1, b = 1
console.log("a : "+a+"   b : "+b);
```

```
x : 4   y : 3
a : 3   b : 3
x : 2   y : 3
a : 1   b : 1
```

비교연산자

- 비교연산자

연산자	이름	설명
>	크다	좌변의 피연산자 보다 우변의 피연산자가 크면 참을 반환한다.
<	작다	좌변의 피연산자 보다 우변의 피연산자가 작으면 참을 반환한다.
>=	크거나 같다	좌변의 피연산자 보다 우변의 피연산자가 크거나 같으면 참을 반환한다.
<=	작거나 같다	좌변의 피연산자 보다 우변의 피연산자가 작거나 같으면 참을 반환합니다.
==	동등	피연산자들이 같으면 참을 반환한다.
!=	부등	피연산자들이 다르면 참을 반환한다.
===	일치	피연산자들이 같고 피연산자들의 같은 형태인 경우 참을 반환한다.
!==	불일치	피연산자들이 다르거나 형태가 다른 경우 참을 반환한다.

비교연산자

- 비교연산자
  - 비교 연산은 true 혹은 false로 결과를 나타낸다.
  - 동등 연산자는 만약 피연산자들이 서로 다르면 같은 타입으로 바꾼다, 그 다음 strict 비교를 적용한다.
  - 만약 두 피연산자 둘 다 객체라면, 자바스크립트가 메모리의 같은 객체를 참조 할 때 내부 내용을 비교하며, 두 피연자가 메모리의 같은 객체를 가리킨다면 두 객체를 같다고 한다.

```
console.log(4 > 3); // true
console.log(4 >= 3);  // true
console.log(3 < 4);  // true
console.log(3 <= 4);  // true
```

```
console.log(1 == 1);          // true
console.log("1" ==  1);          // true
console.log(1 == '1');        // true
console.log(0 == false);      // true
console.log(0 == null);       // false
console.log(0 == undefined); // false
console.log(null  == undefined); // true
```

비교연산자

- 비교연산자
  - 부등 연산자는 피연산자들이 같지 않는다면 참을 반환한다.
  - 만약 두 피연자가 같은 타입이 아니라면, 자바스크립트가 두 피연산자를 비교하기에 적당한 타입으로 바꾼다.
  - 만약 두 피연자가 객체라면, 자바스크립트가 내부 내용을 비교하며, 메모리의 다른 객체를 참조 할 때 다르다고 하며 참을 반환한다.

```
console.log(1 != 2);      // true
console.log(1 != "1");     // false
console.log(1 != '1');     // false
console.log(1 != true);    // false
console.log(0 != false);   // false
```

비교연산자

- 비교연산자
  - 일치 연산자는 피연산자들이 타입 변환 없이 strictly equal일 때를 말한다.
  - 불일치 연산자는 같은 타입에서 값이 다르거나 다른 타입인 경우 참을 반환한다.

```
console.log(3 === 3);    // true
console.log(3 === '3'); // false
console.log(3 !== '3'); // true
console.log(4 !== 3);    // true
```

논리연산자

- 논리연산자

연산자	구문	설명
논리 AND (&&)	expr1 && expr2	expr1을 true로 변환할 수 있는 경우 expr2을 반환하고, 그렇지 않으면 expr1을 반환한다.
논리 OR (||)	expr1 || expr2	expr1을 true로 변환할 수 있으면 expr1을 반환하고, 그렇지 않으면 expr2를 반환한다.
논리 NOT (!)	!expr	단일 피연산자를 true로 변환할 수 있으면 false를 반환합니다. 그렇지 않으면 true를 반환한다.

논리연산자

- 논리연산자

```
console.log(true && true);         // t && t returns true
console.log(true && false);        // t && f returns false
console.log(false && true);         // f && t returns false
console.log(false && (3 == 4));    // f && f returns false
console.log('Cat' && 'Dog');        // t && t returns "Dog"
console.log(false && 'Cat');        // f && t returns false
console.log('Cat' && false);        // t && f returns false
console.log('' && false);       // f && f returns ""
console.log(false && '');           // f && f returns false
```

```
console.log(true || true);         // t || t returns true
console.log(false || true);         // f || t returns true
console.log(true || false);        // t || f returns true
console.log(false || (3 == 4));    // f || f returns false
console.log('Cat' || 'Dog');        // t || t returns "Cat"
console.log(false || 'Cat');        // f || t returns "Cat"
console.log('Cat' || false);        // t || f returns "Cat"
console.log('' || false);       // f || f returns false
console.log(false || '');           // f || f returns ""
varObject = {};
console.log(false || varObject); // f || object returns varObject
```

```
console.log(n1 = !true);                // !t returns false
console.log(n2 = !false);               // !f returns true
console.log(n3 = !'');                  // !f returns true
console.log(n4 = !'Cat');               // !t returns false
```

논리연산자

- 논리연산자
  - NOT 연산자 다수를 연속해서 사용하면 아무 값이나 불리언 원시 값으로 강제 변환할 수 있다.

```
console.log(!!true);                    // !!truthy returns true
console.log(!!{});                      // !!truthy returns true: any object is truthy...
console.log(!!(new Boolean(false)));    // ...even Boolean objects with a false .valueOf()!
console.log(!!false);                   // !!falsy returns false
console.log(!!"");                      // !!falsy returns false
console.log(!!Boolean(false));          // !!falsy returns false
```

```
true
true
true
false
false
false
```

논리연산자

- 불리언 변환 규칙

```
var bCondition1 = 4>3;
var bCondition2 = 1<5;
// 불리언 계산에서, 다음 두 코드는 항상 같다.
console.log(bCondition1 && bCondition2);
console.log(!(!bCondition1 || !bCondition2));
// 불리언 계산에서, 다음 두 코드는 항상 같다.
console.log(bCondition1 || bCondition2);
console.log(!(!bCondition1 && !bCondition2));
// 불리언 계산에서, 다음 두 코드는 항상 같다.
var bCondition = true;
console.log(!!bCondition);
console.log(bCondition);
```

논리연산자

- 단락 규칙
  - && 문 좌측이 거짓일 경우 우측의 연산을 수행하지 않는다.
  - || 문 좌측이 참일 경우 우측의 연산을 수행하지 않는다.

```
var a = 3;
var b = 5;

console.log( ((b++) < 3)&&((a++) < 5) );
console.log(a); //3
console.log(b); //6
console.log( ((b++)>1) || ((a++) < 5) );
console.log(a); //3
console.log(b); //7
```

비트연산자

- 비트연산자
  - 숫자를 2진수화 하여 연산을 할 경우 사용하는 연산자.

연산자	사용방법	설명
비트 AND	a & b	피연산자를 비트로 바꿨을 때 각각 대응하는 비트가 모두 1이면 그 비트값에 1을 반환.
비트 OR	a | b	피연산자를 비트로 바꿨을 때 각각 대응하는 비트가 모두 1이거나 한 쪽이 1이면 1을 반환.
비트 XOR	a ^ b	피연산자를 비트로 바꿨을 때 대응하는 비트가 서로 다르면 1을 반환.
비트 NOT	~ a	피연산자의 반전된 값을 반환.
왼쪽 시프트	a << b	왼쪽으로 숫자를 b만큼 시프트 시킨다
부호 유지 오른쪽 시프트	a >> b	오른쪽으로 숫자를 b만큼 시프트 시킨다(부호 유지)
부호 버림 오른쪽 시프트	a >>> b	오른쪽으로 숫자를 b만큼 시프트 시킨다(부호 유지)

비트연산자

- 비트연산자
  - 오른쪽 시프트 연산에서 >> 는 부호를 유지하지만 >>> 는 부호를 유지하지 않고 0으로 무조건 채워버린다.

```
console.log(10 & 12); // 8
console.log(10 | 12); // 14
console.log(10 ^ 12); // 6
console.log(8 >> 2); // 2
console.log(~10); // -11
console.log(8 << 1); // 16
console.log(-8 >> 2); // -2
console.log(-8 >>> 2);  // ?
```

할당연산자

- 할당연산자
  - 할당 연산자는 오른쪽 피연산자의 값을 왼쪽 피연산자에 할당한다.
  - 기본적인 할당 연산자는 오른쪽의 피연산자 값을 왼쪽 피연산자 값에 할당하는 등호(=) 이다. 즉 x = y 는 y 값을 x에 할당한다.

이름	복합 할당 연산자	뜻	이름	복합 할당 연산자	뜻
할당	x = y	x = y	왼쪽 이동 연산 할당	x <<= y	x = x << y
덧셈 할당	x += y	x = x + y	오른쪽 이동 연산 할당	x >>= y	x = x >> y
뺄셈 할당	x -= y	x = x - y	부호 없는 오른쪽 이동 연산 할당	x >>>= y	x = x >>> y
곱셈 할당	x *= y	x = x * y	비트 AND 할당	x &= y	x = x & y
나눗셈 할당	x /= y	x = x / y	비트 XOR 할당	x ^= y	x = x ^ y
나머지 연산 할당	x %= y	x = x % y	비트 OR 할당	x |= y	x = x | y
지수 연산 할당	x **= y	x = x ** y			

할당연산자

- 할당연산자
  - 할당 : 단순 할당 연산자는 값을 변수에 할당한다. 할당 연산자는 할당의 결과값으로 평가된다. 할당 연산자를 연속해서 사용해, 다수의 변수에 하나의 값을 한꺼번에 할당할 수 있다.

```
x = 5;
y = 10;
z = 25;
console.log(x);
console.log(y);
console.log(z);

x = y;      // x는 10
console.log(x);
console.log(y);
x = y = z; // x, y, z 모두 25
console.log(x);
console.log(y);
console.log(z);
```

할당연산자

- 할당연산자
  - 덧셈할당 : 덧셈 할당 연산자는 변수에 오른쪽 피연산자의 값을 더하고, 그 결과를 변수에 할당한다. 두 피연산자의 자료형이 덧셈 할당 연산자의 행동을 결정한다. 덧셈 연산자처럼 합 또는 연결이 가능하다.

```
// 다음과 같은 변수를 가정
foo = "foo"
bar = 5
baz = true

// Number + Number -> 합
bar += 2 // 7
console.log(bar);
// Boolean + Number -> 합
baz += 1 // 2
console.log(baz);
// Boolean + Boolean -> 합
baz += false // 2
console.log(baz);
// Number + String -> 연결
bar += 'foo' // "5foo"
console.log(bar);
// String + Boolean -> 연결
foo += false // "foofalse"
console.log(foo);
// String + String -> 연결
foo += 'bar' // "foofalsebar"
console.log(foo);
```

할당연산자

- 할당연산자
  - 뺄셈/곱셈/나눗셈/나머지/거듭제곱 할당

```
// 뺄셈
bar = 5;

bar -= 2;     // 3
console.log(bar);
bar -= "foo"; // NaN
console.log(bar);

// 곱셈
bar = 5

bar *= 2     // 10
console.log(bar);
bar *= "foo" // NaN
console.log(bar);

// 나눗셈
bar = 5

bar /= 2     // 2.5
console.log(bar);
bar /= "foo" // NaN
console.log(bar);
bar = 5
bar /= 0     // Infinity
console.log(bar);
```

```
// 나머지 연산
bar = 5

bar %= 2     // 1
console.log(bar);
bar %= "foo" // NaN
console.log(bar);
bar = 5
bar %= 0     // NaN
console.log(bar);

// 거듭제곱 할당
bar = 5

bar **= 2     // 25
console.log(bar);
bar **= "foo" // NaN
console.log(bar);
```

할당연산자

- 할당연산자
  - 비트연산 할당연산자

```
// 왼쪽 시프트 연산
var bar = 5; //  (00000000000000000000000000000101)
bar <<= 2; // 20
console.log(bar);

// 오른쪽 시프트 할당
var bar = 5; //   (00000000000000000000000000000101)
bar >>= 2;   // 1 (00000000000000000000000000000001)
console.log(bar);
var bar = -5; //    (-00000000000000000000000000000101)
bar >>= 2;  // -2 (-00000000000000000000000000000010)
console.log(bar);

// 부호 없는 오른쪽 시프트 할당
var bar = 5; //   (00000000000000000000000000000101)
bar >>>= 2;  // 1 (00000000000000000000000000000001)
console.log(bar);
var bar = -5; // (-00000000000000000000000000000101)
bar >>>= 2; // 1073741822 (00111111111111111111111111111110)
console.log(bar);
```

```
// 비트 AND 할당
var bar = 5;
// 5:     00000000000000000000000000000101
// 2:     00000000000000000000000000000010
bar &= 2; // 0
console.log(bar);

// 비트 XOR 할당
var bar = 5;
bar ^= 2; // 7
// 5: 00000000000000000000000000000101
// 2: 00000000000000000000000000000010
// ----------------------------------
// 7: 00000000000000000000000000000111
console.log(bar);

// 비트 OR 할당
var bar = 5;
bar |= 2; // 7
// 5: 00000000000000000000000000000101
// 2: 00000000000000000000000000000010
// ----------------------------------
// 7: 00000000000000000000000000000111
console.log(bar);
```

쉼표연산자

- 쉼표연산자
  - 단일 표현식을 요구하는 곳에 복수의 표현식을 사용하고 싶을 때 쉼표 연산자를 사용할 수 있다.
  - 가장 흔히 사용되는 곳은 for 반복문에 다수의 매개변수를 제공할 때 쓰인다.
  - 쉼표 연산자는 배열, 객체, 함수의 매개변수와 호출 인수에서 사용하는 쉼표와는 전혀 다르다.

```
var a, b, c;

a = b = 3, c = 4; // 콘솔에는 4를 반환
console.log(a); // 3 (제일 왼쪽)

var x, y, z;

x = (y = 5, z = 6); // 콘솔에는 6을 반환
console.log(x); // 6 (제일 오른쪽)
```

단항연산자

- typeof 연산자
  - typeof 연산자는 피연산자의 타입을 나타내는 문자열을 반환한다.
  - 피연산자 는 어떤 타입인지 반환될 문자열, 변수, 키워드,또는 객체가 된다

```
var myFun = new Function("5 + 2");
var shape = "round";
var size = 1;
var foo = ['Apple', 'Mango', 'Orange'];
var today = new Date();

console.log(typeof myFun);     // returns "function"
console.log(typeof shape);     // returns "string"
console.log(typeof size);      // returns "number"
console.log(typeof foo);       // returns "object"
console.log(typeof today);     // returns "object"
console.log(typeof dontExist); // returns "undefined"
console.log(typeof true); // returns "boolean"
console.log(typeof null); // returns "object"
console.log(typeof Date);     // returns "function"
```

삼항(조건)연산자

- 삼항(조건)연산자
  - 참이냐 거짓인가에 따라 값을 다르게 주는 연산자
  - 항이 3개라서 삼항이라 부르기도 하고 조건에 따라 연산을 달리 해주기 때문에 조건 연산자라 부르기도 한다.
  - 삼항 연산자는 아래와 같이 쓸 수 있다.

(조건식) ? (참일때의 값) : (거짓일때의 값)

```
var age = 29;
var canDrinkAlcohol = (age > 19) ? "True, over 19" : "False, under 19";
console.log(canDrinkAlcohol); // "True, over 19"
```

연산자 우선 순위

- 연산자 우선 순위

Operator type	Individual operators
맴버 연산자	. []
객체 호출/생성 연산자	() new
부정/증가 연산자	! ~ - + ++ -- typeof void delete
곱셈/나눗셈 연산자	* / %
덧셈/뺄셈 연산자	+ -
비트단위 시프트 연산자	<< >> >>>
관계 연산자	< <= > >= in instanceof
같음 비교 연산자	== != === !==
비트 단위 논리곱 연산자	&
비트단위 배타적 논리합 연산자	^
비트단위 논리합 연산자	|
논리 곱 연산자	&&
논리 합 연산자	||
조건 연산자	?:
할당 연산자	= += -= *= /= %= <<= >>= >>>= &= ^= |=
콤마 연산자	,

Null 병합 연산자

- Null 병합 연산자 “??”
  - 여러 피연산자 중 그 값이 ‘확정되어 있는’ 변수를 찾는 연산자

(변수1) ?? (변수2)

  - “ a ?? b “ 라고 할 경우 a라는 변수가 null 이 아니거나 undefined가 아닐 경우 a값을, 아니면 b 값을 출력하는 연산을 한다.
  - 삼항 연산자로 표현할 경우 수식이 복잡해지므로 간단하게 표현하고자 할 경우 위의 연산을 사용한다.

Null 병합 연산자

- Null 병합 연산자 "??"
  - 삼항 연산자와 null 병합 연산자의 비교

```
let a = null;
let b = 3;

x = (a !== null && a !== undefined) ? a : b;

console.log(x); // 3

y = a ?? b;

console.log(y); // 3
```

Null 병합 연산자

- Null 병합 연산자 “??”
  - null 병합 연산자 연속 사용

```
let firstName = null;
let lastName = null;
let nickName = "바이올렛";

// null이나 undefined가 아닌 첫 번째 피연산자
alert(firstName ?? lastName ?? nickName ?? "익명의 사용자"); // 바이올렛
```

Null 병합 연산자

- Null 병합 연산자 “??” 와 “||” 의 차이점
  - ||는 첫 번째 truthy 값을 반환한다.
  - ??는 첫 번째 정의된(defined) 값을 반환한다.

```
let height = 0;

alert(height || 100); // 100
alert(height ?? 100); // 0
```