7. 연산자

연산자(operator)는 하나 이상의 표현식을 대상으로 산술, 할당, 비교, 논리, 타입, 지수 연산(operation) 등을 수행해 하나의 값을 만든다. 이때 연산의 대상을 피연산자(operand)라 한다. 피연산자는 값으로 평가될 수 있는 표현식이어야 한다. 그리고 피연산자와 연산자의 조합으로 이루어진 연산자 표현식도 값으로 평가될 수 있는 표현식이다.

JAVASCRIPT

```
// 산술 연산자
5 * 4 // → 20
// 문자열 연결 연산자
'My name is ' + 'Lee' // → 'My name is Lee'
// 할당 연산자
color = 'red' // → 'red'
// 비교 연산자
3 > 5 // → false
// 논리 연산자
true && false // → false
// 타입 연산자
typeof 'Hi' // → string
```

피연산자가 “값”이라는 명사의 역할을 한다면 연산자는 “피연산자를 연산하여 새로운 값을 만든다”라는 동사의 역할을 한다고 볼 수 있다. 다시 말해, 피연산자는 연산의 대상이 되어야 하므로 값으로 평가할 수 있어야 한다. 연산자는 값으로 평가된 피연산자를 연산해 새로운 값을 만든다.

자바스크립트가 제공하는 다양한 연산자에 대해 살펴보도록 하자.

1. 산술 연산자

산술 연산자(arithmetic operator)는 피연산자를 대상으로 수학적 계산을 수행해 새로운 숫자 값을 만든다. 산술 연산이 불가능한 경우, NaN을 반환한다.

산술 연산자는 피연산자의 개수에 따라 이항 산술 연산자와 단항 산술 연산자로 구분할 수 있다.

1.1. 이항 산술 연산자

이항 산술 연산자는 2개의 피연산자를 산술 연산하여 숫자 타입의 값을 만든다.

모든 이항 산술 연산자는 피연산자의 값을 변경하는 부수 효과(side effect)가 없다. 다시 말해 어떤 산술 연산을 해도 피연산자의 값이 바뀌는 경우는 없다. 언제나 새로운 값을 만들 뿐이다.

이항 산술 연산자	의미	부수 효과
+	덧셈	×
-	뺄셈	×
*	곱셈	×
/	나눗셈	×
%	나머지	×

JAVASCRIPT

```
5 + 2; // → 7
5 - 2; // → 3
5 * 2; // → 10
5 / 2; // → 2.5
5 % 2; // → 1
```

1.2. 단항 산술 연산자

단항(unary) 산술 연산자는 1개의 피연산자를 산술 연산하여 숫자 타입의 값을 만든다.

단항 산술 연산자	의미	부수 효과
++	증가	○
--	감소	○
+	어떠한 효과도 없다. 음수를 양수로 반전하지도 않는다.	×
-	양수를 음수로 음수를 양수로 반전한 값을 반환한다.	×

주의할 것은 이항 산술 연산자와는 달리 **증가/감소(++/–) 연산자는 피연산자의 값을 변경하는 부수 효과가 있다**. 다시 말해 증가/감소 연산을 하면 피연산자의 값을 변경하는 암묵적 할당이 이루어진다.

JAVASCRIPT

```
var x = 1;

// ++ 연산자는 피연산자의 값을 변경하는 암묵적 할당이 이루어진다.
x++; // x = x + 1;
console.log(x); // 2

// -- 연산자는 피연산자의 값을 변경하는 암묵적 할당이 이루어진다.
x--; // x = x - 1;
console.log(x); // 1
```

증가/감소(++/--) 연산자는 위치에 의미가 있다.

- 피연산자 앞에 위치한 전위 증가/감소 연산자(prefix increment/decrement operator)는 먼저 피연산자의 값을 증가/감소시킨 후, 다른 연산을 수행한다.
- 피연산자 뒤에 위치한 후위 증가/감소 연산자(postfix increment/decrement operator)는 먼저 다른 연산을 수행한 후, 피연산자의 값을 증가/감소시킨다.

JAVASCRIPT

```
var x = 5, result;

// 선할당 후증가 (postfix increment operator)
result = x++;
console.log(result, x); // 5 6
```

```
// 선증가 후할당 (prefix increment operator)
result = ++x;
console.log(result, x); // 7 7

// 선할당 후감소 (postfix decrement operator)
result = x--;
console.log(result, x); // 7 6

// 선감소 후할당 (prefix decrement operator)
result = --x;
console.log(result, x); // 5 5
```

+ 단항 연산자는 피연산자에 어떠한 효과도 없다. 음수를 양수로 반전하지도 않는다. 그런데 숫자 타입이 아닌 피연산자에 사용하면 피연산자를 숫자 타입으로 변환하여 반환한다. 이때 피연산자를 변경하는 것은 아니고 숫자 타입으로 변환한 값을 생성해서 반환한다. 따라서 부수 효과는 없다.

JAVASCRIPT

```
// 아무런 효과가 없다.
+10;    // → 10
+(-10); // → -10

// 문자열을 숫자로 타입 변환한다.
+'10'; // → 10

// 불리언 값을 숫자로 타입 변환한다.
+true; // → 1

// 불리언 값을 숫자로 타입 변환한다.
+false; // → 0

// 문자열을 숫자로 타입 변환할 수 없으므로 NaN을 반환한다.
+'Hello'; // → NaN
```

- 단항 연산자는 피연산자의 부호를 반전한 값을 반환한다. + 단항 연산자와 마찬가지로 숫자 타입이 아닌 피연산자에 사용하면 피연산자를 숫자 타입으로 변환하여 반환한다. 이때 피연산자를 변경하는 것은 아니고 부호를 반전한 값을 생성해서 반환한다. 따라서 부수 효과는 없다.

JAVASCRIPT

```
// 부호를 반전한다.
-(-10); // → 10

// 문자열을 숫자로 타입 변환한다.
-'10'; // → -10

// 불리언 값을 숫자로 타입 변환한다.
-true; // → -1

// 문자열을 숫자로 타입 변환할 수 없으므로 NaN을 반환한다.
-'Hello'; // → NaN
```

1.3. 문자열 연결 연산자

+ **연산자는 피연산자 중 하나 이상이 문자열인 경우, 문자열 연결 연산자로 동작한다.** 그 외의 경우는 산술 연산자로 동작한다. 아래 예제를 살펴보자.

JAVASCRIPT

```
// 문자열 연결 연산자
'1' + 2; // → '12'
1 + '2'; // → '12'

// 산술 연산자
1 + 2; // → 3

// true는 1로 타입 변환된다.
1 + true; // → 2

// false는 0으로 타입 변환된다.
1 + false; // → 1

// null는 0으로 타입 변환된다.
1 + null; // → 1

// undefined는 숫자로 타입 변환되지 않는다.
```

```
+undefined;    // → NaN
1 + undefined; // → NaN
```

이 예제에서 주목할 것은 개발자의 의도와는 상관없이 자바스크립트 엔진에 의해 암묵적으로 타입이 자동 변환되기도 한다는 것이다. 위 예제에서 `1 + true` 를 연산하면 자바스크립트 엔진은 암묵적으로 불리언 타입의 값인 true를 숫자 타입인 1로 타입을 강제 변환한 후 연산을 수행한다.

이를 **암묵적 타입 변환**(implicit coercion) **또는 타입 강제 변환**(type coercion)이라고 한다. 앞서 살펴본 +/- 단항 연산자도 암묵적 타입 변환이 발생한 것이다. 이에 대해서는 "9. 타입 변환과 단축 평가"에서 자세히 살펴볼 것이다.

#2. 할당 연산자

할당 연산자(assignment operator)는 우항에 있는 피연산자의 평가 결과를 좌항에 있는 변수에 할당한다. 할당 연산자는 좌항의 변수에 값을 할당하므로 변수의 값이 변하는 부수 효과가 있다.

할당 연산자	사례	동일 표현	부수 효과
=	x = 5	x = 5	○
+=	x += 5	x = x + 5	○
-=	x -= 5	x = x - 5	○
*=	x *= 5	x = x * 5	○
/=	x /= 5	x = x / 5	○
%=	x %= 5	x = x % 5	○

JAVASCRIPT

```
var x;

x = 10;
console.log(x); // 10

x += 5; // x = x + 5;
console.log(x); // 15
```

```
x -= 5; // x = x - 5;
console.log(x); // 10

x *= 5; // x = x * 5;
console.log(x); // 50

x /= 5; // x = x / 5;
console.log(x); // 10

x %= 5; // x = x % 5;
console.log(x); // 0

var str = 'My name is ';

// 문자열 연결 연산자
str += 'Lee'; // str = str + 'Lee';
console.log(str); // 'My name is Lee'
```

표현식은 값으로 평가된다고 하였다. 그렇다면 할당 연산은 표현식인 문일까? 아래의 예제를 살펴보자.

JAVASCRIPT

```
var x;

// 할당문은 표현식인 문이다.
console.log(x = 10); // 10
```

할당문은 변수에 값을 할당하는 부수 효과만 있을 뿐 값으로 평가되지 않을 것처럼 보인다. 하지만 할당문은 값으로 평가되는 표현식인 문이다. 할당문은 할당된 값으로 평가된다. 위 예제의 할당문 `x = 10`은 x에 할당된 숫자 값 10으로 평가된다. 따라서 아래와 같이 할당문을 다른 변수에 할당할 수도 있다. 이러한 특징을 활용해 여러 변수에 동일한 값을 연쇄 할당할 수 있다.

JAVASCRIPT

```
var a, b, c;

// 연쇄 할당. 오른쪽에서 왼쪽으로 진행.
// ① c = 0 : 0으로 평가된다
```

```
// ② b = 0 : 0으로 평가된다
// ③ a = 0 : 0으로 평가된다
a = b = c = 0;

console.log(a, b, c); // 0 0 0
```

3. 비교 연산자

비교 연산자(comparison operator)는 좌항과 우항의 피연산자를 비교한 다음 그 결과를 불리언 값으로 반환한다. 비교 연산자는 if 문이나 for 문과 같은 제어문의 조건식에서 주로 사용한다.

3.1. 동등 / 일치 비교 연산자

동등 비교(loose equality) 연산자와 일치 비교(strict equality) 연산자는 좌항과 우항의 피연산자가 같은 값으로 평가되는지 비교하여 불리언 값을 반환한다. 하지만 비교하는 엄격성의 정도가 다르다. 동등 비교 연산자는 느슨한 비교를 하지만 일치 비교 연산자는 엄격한 비교를 한다.

비교 연산자	의미	사례	설명	부수 효과
==	동등 비교	x == y	x와 y의 값이 같음	×
===	일치 비교	x === y	x와 y의 값과 타입이 같음	×
!=	부동등 비교	x != y	x와 y의 값이 다름	×
!==	불일치 비교	x !== y	x와 y의 값과 타입이 다름	×

"7.1.3. 문자열 연결 연산자"에서 언급했듯이 개발자의 의도와는 상관없이 자바스크립트 엔진에 의해 암묵적으로 타입이 자동 변환되기도 한다. 이를 암묵적 타입 변환이라 부른다고 했다.

동등 비교(==) 연산자는 좌항과 우항의 피연산자를 비교할 때 먼저 암묵적 타입 변환을 통해 타입을 일치시킨 후, 같은 값인지 비교한다. 따라서 동등 비교 연산자는 좌항과 우항의 피연산자가 타입은 다르더라도 암묵적 타입 변환 후에 같은 값일 수 있다면 true를 반환한다.

JAVASCRIPT

```
// 동등 비교
5 == 5; // → true

// 타입은 다르지만 암묵적 타입 변환을 통해 타입을 일치시키면 동등하다.
5 == '5'; // → true
```

결론부터 말하자면 동등 비교 연산자는 편리한 경우도 있지만 결과를 예측하기 어렵기 때문에 부작용을 일으킬 수 있으므로 사용하지 않는 편이 좋다. 아래 예제를 살펴보자.

JAVASCRIPT

```
// 동등 비교. 결과를 예측하기 어렵다.
'0' == ''; // → false
0 == '';   // → true
0 == '0';  // → true
false == 'false';   // → false
false == '0';       // → true
false == null;      // → false
false == undefined; // → false
```

위 예제와 같은 코드를 작성할 개발자는 드물겠지만 이처럼 동등 비교(==) 연산자는 예측하기 어려운 결과를 만들어낸다. 위 예제는 이해하려 하지 않아도 된다. 다만 동등 비교 연산자를 사용하지 말고 일치 비교 연산자를 사용하면 된다.

일치 비교(===) 연산자는 좌항과 우항의 피연산자가 타입도 같고 값도 같은 경우에 한하여 true를 반환한다. 다시 말해 암묵적 타입 변환을 하지 않고 값을 비교한다.

JAVASCRIPT

```
// 일치 비교
5 === 5; // → true

// 암묵적 타입 변환을 하지 않고 값을 비교한다.
// 즉, 값과 타입이 모두 같은 경우만 true를 반환한다.
5 === '5'; // → false
```

일치 비교 연산자는 예측하기 쉽다. 위에 살펴본 동등 비교 연산자의 해괴망측한 예제는 모두 false를 반환한다.

일치 비교 연산자에서 주의할 것은 NaN이다.

JAVASCRIPT

```
// NaN은 자신과 일치하지 않는 유일한 값이다.
NaN === NaN; // → false
```

NaN은 자신과 일치하지 않는 유일한 값이다. 따라서 숫자가 NaN인지 조사하려면 빌트인 함수 isNaN을 사용한다

JAVASCRIPT

```
// 빌트인 함수 isNaN은 주어진 값이 NaN인지 체크하고 그 결과를 반환한다.
isNaN(NaN); // → true
isNaN(10);  // → false
isNaN(1 + undefined); // → true
```

숫자 0도 주의하도록 하자. 자바스크립트에는 양의 0과 음의 0이 있는데 이들을 비교하면 true를 반환한다.

JAVASCRIPT

```
// 양의 0과 음의 0의 비교. 일치 비교/동등 비교 모두 true이다.
0 === -0; // → true
0 == -0;  // → true
```

Object.is **메소드**

위에서 살펴본 바와 같이 동등 비교 연산자(==)와 일치 비교 연산자(===)는 +0과 -0을 동일하다고 평가한다. 또한 동일한 값인 NaN과 NaN을 비교하면 다른 값이라고 평가한다.

ES6에서 새롭게 도입된 Object.is 메소드는 아래와 같이 예측 가능한 정확한 비교 결과를 반환한다. 그 외에는 일치 비교 연산자(===)와 동일하게 동작한다.

JAVASCRIPT

```
-0 === +0;          // → true
Object.is(-0, +0); // → false

NaN === NaN;        // → false
Object.is(NaN, NaN); // → true
```

부동등 비교 연산자(!=)와 불일치 비교 연산자(!==)는 동등 비교(==) 연산자와 일치 비교(===) 연산자의 반대 개념이다.

JAVASCRIPT

```
// 부동등 비교
5 ≠ 8;   // → true
5 ≠ 5;   // → false
5 ≠ '5'; // → false

// 불일치 비교
5 !== 8;   // → true
5 !== 5;   // → false
5 !== '5'; // → true
```

3.2. 대소 관계 비교 연산자

대소 관계 비교 연산자는 피연산자의 크기를 비교하여 불리언 값을 반환한다.

대소 관계 비교 연산자	예제	설명	부수 효과
>	x > y	x가 y보다 크다	X
<	x < y	x가 y보다 작다	X
>=	x >= y	x가 y보다 같거나 크다	X
<=	x <= y	x가 y보다 같거나 크다	X

JAVASCRIPT

```
// 대소 관계 비교
5 > 0;  // → true
5 > 5;  // → false
5 ⩾ 5; // → true
5 ⩽ 5; // → true
```

4. 삼항 조건 연산자

삼항 조건 연산자(ternary operator)는 조건식의 평가 결과에 따라 반환할 값을 결정한다. 자바스크립트의 유일한 삼항 연산자이며 부수 효과는 없다. 삼항 조건 연산자 표현식은 아래와 같이 사용한다.

CODE

```
조건식 ? 조건식이 true일때 반환할 값 : 조건식이 false일때 반환할 값
```

물음표(?) 앞의 첫번째 피연산자는 조건식, 즉 불리언 타입의 값으로 평가될 표현식이다. 만약 조건식의 평가 결과가 불리언 값이 아니면 불리언 값으로 암묵적 타입 변환된다. 이때 조건식이 참이면 콜론(:) 앞의 두번째 피연산자가 평가되어 반환되고, 거짓이면 콜론(:) 뒤의 세번째 피연산자가 평가되어 반환된다.

JAVASCRIPT

```
var x = 2;

// 2 % 2는 0이고 0은 false로 암묵적 타입 변환된다.
var result = x % 2 ? '홀수' : '짝수';

console.log(result); // 짝수
```

삼항 조건 연산자는 다음 장에서 살펴볼 if...else 문을 사용해도 동일한 처리를 할 수 있다.

JAVASCRIPT

```
var x = 2, result;

// 2 % 2는 0이고 0은 false로 암묵적 타입 변환된다.
```

```
if (x % 2) result = '홀수';
else       result = '짝수';

console.log(result); // 짝수
```

하지만 if...else 문은 표현식이 아닌 문이다. 따라서 if...else 문은 값처럼 사용할 수 없다.

JAVASCRIPT

```
var x = 10;

// if ... else 문은 표현식이 아닌 문이다. 따라서 값처럼 사용할 수 없다.
var result = if (x % 2) { result = '홀수'; } else { result = '짝수'; };
// SyntaxError: Unexpected token if
```

삼항 조건 연산자 표현식은 값으로 평가할 수 있는 표현식인 문이다. 따라서 삼항 조건 연산자식은 값처럼 다른 표현식의 일부가 될 수 있어 매우 유용하다.

JAVASCRIPT

```
var x = 10;

// 삼항 연산자 표현식은 표현식인 문이다. 따라서 값처럼 사용할 수 있다.
var result = x % 2 ? '홀수' : '짝수';
console.log(result); // 짝수
```

조건에 따라 어떤 값을 결정해야 한다면 if...else 문보다 삼항 조건 연산자 표현식을 사용하는 것이 유리하다. 하지만 조건에 따라 수행해야 할 문이 하나가 아니라 여러 개라면 if...else 문이 보다 가독성이 좋다.

5. 논리 연산자

논리 연산자(logical operator)는 우항과 좌항의 피연산자(부정 논리 연산자의 경우, 우항의 피연산자)를 논리 연산한다.

논리 연산자	의미	부수 효과
||	논리합(OR)	X
&&	논리곱(AND)	X
!	부정(NOT)	X

JAVASCRIPT

```
// 논리합(||) 연산자
true || true;   // → true
true || false;  // → true
false || true;  // → true
false || false; // → false

// 논리곱(&&) 연산자
true && true;   // → true
true && false;  // → false
false && true;  // → false
false && false; // → false

// 논리 부정(!) 연산자
!true;  // → false
!false; // → true
```

논리 부정(!) 연산자는 언제나 불리언 값을 반환한다. 단, 피연산자가 반드시 불리언 값일 필요는 없다. 만약 피연산자가 불리언 값이 아니면 불리언 타입으로 암묵적 타입 변환된다.

JAVASCRIPT

```
// 암묵적 타입 변환
!0;       // → true
!'Hello'; // → false
```

논리합(||) 또는 논리곱(&&) 연산자 표현식의 평가 결과는 불리언 값이 아닐 수도 있다. **논리합(||) 또는 논리곱(&&) 연산자 표현식은 언제나 2개의 피연산자 중 어느 한쪽으로 평가된다.**

JAVASCRIPT

```
// 단축 평가
'Cat' && 'Dog'; // → 'Dog'
```

이에 대해서는 "9.4. 단축 평가"에서 자세히 살펴보기로 하자.

드 모르간의 법칙

논리 연산자로 구성된 복잡한 표현식은 가독성이 좋지 않아 한눈에 이해하기 어려울 때가 있다. 이러한 경우, 드 모르간의 법칙을 활용하면 복잡한 표현식을 좀 더 가독성 좋은 표현식으로 변환할 수 있다.

JAVASCRIPT

```
!(x || y) === (!x && !y)
!(x && y) === (!x || !y)
```

#6. 쉼표 연산자

쉼표(,) 연산자는 왼쪽 피연산자부터 차례대로 피연산자를 평가하고 마지막 피연산자의 평가가 끝나면 마지막 피연산자의 평가 결과를 반환한다.

JAVASCRIPT

```
var x, y, z;

x = 1, y = 2, z = 3; // 3
```

#7. 그룹 연산자

그룹 연산자 (...)는 자신의 피연산자인 표현식을 가장 먼저 평가한다. 따라서 그룹 연산자를 사용하면 연산자의 우선 순위를 조절할 수 있다.

JAVASCRIPT

```
10 * 2 + 3; // → 23

// 그룹 연산자를 사용하여 우선 순위 조절
10 * (2 + 3); // → 50
```

위 예제의 첫번째 문은 `10 * 2` 를 먼저 연산하고 그 다음 `20 + 3` 을 연산한다. 수학에서와 마찬가지로 곱셈 연산자 *가 덧셈 연산자 +보다 우선 순위가 높기 때문이다.

두번째 문은 그룹 연산자로 감싼 표현식을 먼저 연산한다. 따라서 `2 + 3` 을 먼저 연산하고 그 다음 `10 * 5` 를 연산한다.

8. typeof 연산자

typeof 연산자는 피연산자의 데이터 타입을 문자열로 반환한다. typeof 연산자는 7가지 문자열 “string”, “number”, “boolean”, “undefined”, “symbol”, “object”, “function” 중 하나를 반환한다. “null”을 반환하는 경우는 없으며 함수의 경우 “function”을 반환한다. 이처럼 typeof 연산자가 반환하는 문자열은 7개의 데이터 타입과 정확히 일치하지는 않는다.

JAVASCRIPT

```
typeof ''               // → "string"
typeof 1                // → "number"
typeof NaN              // → "number"
typeof true             // → "boolean"
typeof undefined        // → "undefined"
typeof Symbol()         // → "symbol"
typeof null             // → "object"
typeof []               // → "object"
typeof {}               // → "object"
typeof new Date()       // → "object"
typeof /test/gi         // → "object"
typeof function () {}   // → "function"
```

typeof 연산자로 null 값을 연산해 보면 “null”이 아닌 “object”를 반환하는 것에 주의하자. 이것은 자바스크립트의 첫 번째 버전의 버그이다. 하지만 기존 코드에 영향을 줄 수 있기 때문에 아직까지 수정되지 못하고 있다.

- “typeof null”의 역사

따라서 null 타입을 확인할 때는 typeof 연산자를 사용하지 말고 일치 연산자(===)를 사용하도록 하자.

JAVASCRIPT

```
var foo = null;

typeof foo === null; // → false
foo === null;        // → true
```

또 하나 주의해야 할 것이 있다. 선언하지 않은 식별자를 typeof 연산자로 연산해 보면 ReferenceError가 발생하지 않고 “undefined”를 반환한다.

JAVASCRIPT

```
// 식별자 undeclared는 선언한 적이 없다.
typeof undeclared; // → undefined
```

typeof 연산자가 선언하지 않은 식별자를 연산했을 때 “undefined”를 반환하는 것을 카일 심슨의 “You don’t know JS”에서는 특별한 안전 가드(safety guard)로 설명한다. 하지만 모던 자바스크립트 개발에서는 대부분 모듈을 사용하고 전역 변수인 플래그를 사용하지 않으므로 의도적으로 사용할 필요는 없다.

#9. 지수 연산자

ES7에서 새롭게 도입된 지수 연산자는 좌항의 피연산자를 밑으로, 우항의 피연산자를 지수로 거듭 제곱하여 숫자 타입의 값을 반환한다.

JAVASCRIPT

```
2 ** 2;   // → 4
2 ** 2.5; // → 5.65685424949238
```

```
2 ** 0;   // → 1
2 ** -2;  // → 0.25
```

지수 연산자가 도입되기 이전에는 Math.pow 메소드를 사용하였다.

JAVASCRIPT

```
Math.pow(2, 2);   // → 4
Math.pow(2, 2.5); // → 5.65685424949238
Math.pow(2, 0);   // → 1
Math.pow(2, -2);  // → 0.25
```

지수 연산자는 여러 개의 피연산자를 사용할 경우, Math.pow 메소드보다 가독성이 좋다.

JAVASCRIPT

```
2 ** 2 ** 2; // → 16
Math.pow(Math.pow(2, 2), 2); // → 16
```

음수를 거듭제곱의 밑으로 계산하려면 아래와 같이 괄호로 묶어야 한다.

JAVASCRIPT

```
-5 ** 2;
// → SyntaxError: Unary operator used immediately before exponentiation exp
ression. Parenthesis must be used to disambiguate operator precedence
(-5) ** 2; // → 25
```

지수 연산자는 다른 산술 연산자와 마찬가지로 할당 연산자와 함께 사용할 수 있다.

JAVASCRIPT

```
var num = 5;
num **= 2; // → 25
```

지수 연산자는 모든 이항 연산자보다 우선 순위가 높다.

JAVASCRIPT

```
2 * 5 ** 2; // → 50
```

#10. 그 외의 연산자

지금까지 소개한 연산자 이외에도 다양한 연산자가 존재한다. 다만 아직 소개하지 않은 연산자들은 다른 주제와 밀접하게 연결되어 있어 해당 주제를 소개하는 장에서 알아보도록 할 것이다.

연산자	개요	참조
delete	프로퍼티 삭제	10.8. 프로퍼티 삭제
new	생성자 함수를 호출할 때 사용	17.2.6. new 연산자
instanceof	좌변의 객체가 우변의 생성자 함수와 연결된 인스턴스인지 판별	19.11. instanceof 연산자
in	프로퍼티 존재 확인	19.14. 프로퍼티 존재 확인

#11. 연산자의 부수 효과

대부분의 연산자는 다른 코드에 영향을 주지 않는다. 예를 들어, 1 * 2는 다른 코드에 어떠한 영향도 주지 않는다. 하지만 일부 연산자는 다른 코드에 영향을 주는 부수 효과(side effect)가 있다.

부수 효과가 있는 연산자는 할당(=) 연산자, 증가/감소(++/--) 연산자, delete 연산자("10.8. 프로퍼티 삭제" 참고)이다.

JAVASCRIPT

```
var x;

// 할당 연산자는 변수 값이 변하는 부수 효과가 있다.
// 이는 변수 x를 사용하는 다른 코드에 영향을 준다.
x = 1;
console.log(x); // 1
```

```
// 증가/감소(++/--) 연산자는 피연산자의 값을 변경하는 부수 효과가 있다.
// 피연산자 x의 값이 변경된다. 이는 변수 x를 사용하는 다른 코드에 영향을 준다.
x++;
console.log(x); // 2

var o = { a: 1 };

// delete 연산자는 객체의 프로퍼티를 삭제하는 부수 효과가 있다.
// 이는 객체 o를 사용하는 다른 코드에 영향을 준다.
delete o.a;
console.log(o); // {}
```