# Comparison Operator (비교 연산자)

## ☑ Comparison Operator

수학 시간에 아래와 같은 다양한 comparison operator에 대해 학습한 바 있다.
JavaScript에서 기본 수학 연산은 아래와 같은 문법을 사용해 표현할 수 있다.

- ☞ greater/less than: a > b, a < b
- ☞ greater/less than or equals: a >= b, a <= b
- ☞ equals(동등): a == b. 등호 =가 두 개 연달아 오는 것에 유의한다. a = b와 같이 등호가 하나일 때는 assignment(할당)을 의미한다.
- ☞ not equals(부등): 같지 않음을 나타내는 수학 기호 ≠는 JavaScript에선 a != b로 나타낸다. assignment operator = 앞에 느낌표 !를 붙여서 표시한다.

이 장에서는 비교 시 일어나는 기이한 현상을 포함하여 다양한 data type을 대상으로 JavaScript가 어떻게 비교를 하는지에 대해 학습하도록 한다. 마지막에 "JavaScript quirks" 관련 문제를 피하기 위한 좋은 방법을 찾을 수 있다.

### ❖ Boolean is result

다른 operator와 마찬가지로 comparison operator 역시 값을 반환한다. 반환 값은 boolean 이다.

- ☞ true가 반환되면, 'yes', 'correct', 'truth'를 의미한다.
- ☞ false가 반환되면, 'no', 'wrong', 'not the truth'를 의미한다.

**[ 예시 ]**

```
alert( 2 > 1 );  // true
alert( 2 == 1 ); // false
alert( 2 != 1 ); // true
```

반환된 boolean value는 다른 여타 값처럼 변수에 할당 할 수 있다.

```
let result = 5 > 4;       // 비교 결과를 변수에 할당
alert( result );          // true
```

## ❖ String comparison

JavaScript는 'dictionary' 순으로 string을 비교한다. '사전편집(lexicographical)'순 이라고 불리기도 하는 이 기준을 적용하면 사전 뒤쪽의 string은 사전 앞쪽의 string보다 크다고 판단된다.

실제 단어를 사전에 실을 때 단어를 구성하는 문자 하나하나를 비교하여 등재 순서를 정하는 것과 같이 JavaScript도 string을 구성하는 문자 하나하나를 비교해가며 string을 비교한다.

**[ 예시 ]**

```
alert( 'Z' > 'A' );      // true
alert( 'Glow' > 'Glee' ); // true
alert( 'Bee' > 'Be' );    // true
```

string 비교 시 적용되는 algorithm은 다음과 같다.

① 두 string의 첫 글자를 비교한다.
② 첫 번째 string의 첫 글자가 다른 string의 첫 글자보다 크면(작으면), 첫 번째 string이 두 번째 string보다 크다고(작다고) 결론 내고 비교를 종료한다.
③ 두 string의 첫 글자가 같으면 두 번째 글자를 같은 방식으로 비교한다.
④ 글자 간 비교가 끝날 때까지 이 과정을 반복한다.
⑤ 비교가 종료되었고 string의 길이도 같다면 두 string은 동일하다고 결론을 낸다. 비교가 종료되었지만 두 string의 길이가 다르면 길이가 긴 string이 더 크다고 결론을 낸다.

예시의 'Z' > 'A'는 위 algorithm의 첫 번째 단계에서 비교 결과가 도출된다. 반면, string 'Glow'와 'Glee'는 복수의 문자로 이루어진 string이기 때문에, 아래와 같은 순서로 string 비교가 이뤄진다.

① G는 G와 같다.
② l은 l과 같다.
③ o는 e보다 크기 때문에 여기서 비교가 종료되고, o가 있는 첫 번째 string 'Glow'가 더 크다는 결론이 도출된다.

### ⊙ 정확히는 사전순이 아니라 Unicode 순이다.

JavaScript의 string 비교 algorithm은 사전이나 전화번호부에서 사용되는 정렬 algorithm과 아주 유사하지만, 완전히 같진 않다.
차이점 중 하나는 JavaScript는 대·소문자를 따진다는 것이다. 대문자 'A'와 소문자 'a'를 비교했을 때 소문자 'a'가 더 크다. JavaScript 내부에서 사용되는 인코딩 표인 Unicode에선 소문자가 대문자보다 더 큰 index를 갖기 때문이다.

## ❖ Comparison of different types

비교하려는 값의 type이 다르면 JavaScript는 이 값들을 number로 바꾼다.

**[ 예시 ]**

```
alert( '2' > 1 );   // true, string '2'가 숫자 2로 변환된 후 비교가 진행된다.
alert( '01' == 1 ); // true, string '01'이 숫자 1로 변환된 후 비교가 진행된다.
```

boolean value의 경우 true는 1, false는 0으로 변환된 후 비교가 이루어 진다.

**[ 예시 ]**

```
alert( true == 1 );  // true
alert( false == 0 ); // true
```

### ⊙ 흥미로운 상황

동시에 일어나지 않을 법한 두 상황이 동시에 일어나는 경우도 있다.

☞ 동등 비교(==) 시 true를 반환한다.
☞ 논리 평가 시 값 하나는 true, 다른 값 하나는 false를 반환한다.

**[ 예시 ]**

```
let a = 0;
alert( Boolean(a) ); // false

let b = "0";
alert( Boolean(b) ); // true

alert(a == b);       // true!
```

두 값(a와 b)을 비교하면 true가 반환되는데, 값을 논리 평가한 후 비교하면 하나는 true, 하나는 false가 반환된다는 점에 고개를 갸우뚱할 수도 있다. 그런데 JavaScript 관점에선 이런 결과가 아주 자연스럽다. 동등 비교 operator == 는 (예시에서 string "0"을 숫자 0으로 변환시킨 것처럼) operand를 숫자형으로 바꾸지만, 'Boolean'을 사용한 명시적 변환에는 다른 규칙이 사용되기 때문이다.

## ❖ Strict equality

regular equals operator == 은 0과 false를 구별하지 못한다.

```
alert( 0 == false ); // true
```

operand가 empty string일 때도 같은 문제가 발생한다.

```
alert( '' == false ); // true
```

이런 문제는 equals operator == 가 type 다른 operand를 비교할 때 operand를 number로 바꾸기 때문에 발생한다. empty string과 false는 number로 변환하면 0이 된다.

그렇다면 0과 false는 어떻게 구별할 수 있을까?

☞ **strict equality operator === 를 사용하면 type conversion 없이 값을 비교할 수 있다.**

equality operator === operator는 strict equality operator 이다. 자료형의 equals(동등) 여부까지 검사하기 때문에 operand a와 b의 형이 다를 경우 a === b는 즉시 false를 반환한다.

**[ 예시 ]**

```
alert( 0 === false ); // false, operand의 형이 다르기 때문이다.
```

equality operator ===가 equals operator ==의 엄격한 버전인 것처럼, 'non-equality' operator !==는 not equals operator != 의 엄격한 버전이다.

strict equality operator는 equals operator보다 한 글자 더 길긴 하지만, 비교 결과가 명확하기 때문에 에러가 발생할 확률을 줄여준다.

## ❖ null이나 undefined와 비교하기

null이나 undefined를 다른 값과 비교할 땐 예상치 않은 일들이 발생한다. 일단 몇 가지 규칙을 먼저 살펴본 후, 어떤 예상치 않은 일들이 일어나는지 구체적인 예시를 통해 살펴보도록 한다.

☞ **strict equality operator ===를 사용하여 null과 undefined를 비교**

두 값의 자료형이 다르기 때문에 equality 비교 시 false가 반환된다.

```
alert( null === undefined );      // false
```

☞ **equals operator ==를 사용하여 null과 undefined를 비교**

equals operator를 사용해 null과 undefined를 비교하면 특별한 규칙이 적용돼 true가 반환된다. equals operator는 null과 undefined를 '각별한 커플'처럼 취급한다. 두 값은 자기들끼리는 잘 어울리지만 다른 값들과는 잘 어울리지 못한다.

```
alert( null == undefined ); // true
```

☞ **maths operator나 기타 비교 operator <, >, <=, >=를 사용하여 null과 undefined를 비교**

null과 undefined는 number로 변환된다. null은 0, undefined는 NaN으로 변한다.

이제 위에서 살펴본 3 가지 규칙들이 어떤 흥미로운 edge case를 만들어내는지 알아보자. 이후, 어떻게 하면 edge case가 만들어내는 함정에 빠지지 않을 수 있을지에 대해 알아보자.

### ⊙ Strange result: null vs 0

null과 0을 비교해 보자.

```
alert( null > 0 );  // (1) false
alert( null == 0 ); // (2) false
alert( null >= 0 ); // (3) true
```

위 비교 결과는 논리에 맞지 않아 보인다. (3)에서 null은 0보다 크거나 같다고 했기 때문에, (1)이나 (2) 중 하나는 true 이어야 하는데 둘 다 false을 반환하고 있다.
이런 결과가 나타나는 이유는 equals operator ==와 기타 comparison operator <, >, <=, >=의 동작 방식이 다르기 때문이다. (1)에서 null > 0이 false를, (3)에서 null >= 0이 true를 반환하는 이유는 (기타 comparison operator의 동작 원리에 따라) null이 number로 변환돼 0이 되기 때문이다.

그런데 equals operator ==는 operand가 undefined나 null일 때 형 변환을 하지 않는다. undefined와 null을 비교하는 경우에만 true를 반환하고, 그 이외의 경우(null이나 undefined를 다른 값과 비교할 때)는 무조건 false를 반환한다. 이런 이유 때문에 (2)는 false를 반환한다.

### ⊙ 비교가 불가능한 undefined

undefined를 다른 값과 비교해서는 안 된다.

```
alert( undefined > 0 );  // false (1)
alert( undefined < 0 );  // false (2)
alert( undefined == 0 ); // false (3)
```

위 예시를 보면 undefined는 0을 매우 싫어하는 것처럼 보인다. 항상 false를 반환하고 있다.

이런 결과는 아래와 같은 이유 때문에 발생한다.
(1)과 (2)에선 undefined가 NaN으로 변환되는데(number로의 변환), NaN이 operand인 경우 comparison operator는 항상 false를 반환한다. undefined는 null이나 undefined와 같고, 그 이외의 값과는 같지 않기 때문에 (3)은 false를 반환한다.

## ⊙ 함정 피하기

위와 같은 edge case를 왜 살펴보았을까? 이런 예외적인 경우를 꼭 기억해 놓고 있어야만 할까? 그렇지는 않다. 개발을 하다 보면 자연스레 이런 경우를 만나고 점차 익숙해지기 때문에 지금 당장 암기해야 할 필요는 없다. 그러나 아래와 같은 방법을 사용해 이런 예외 상황을 미리 예방할 수 있다는 점은 알아두길 권장한다.

- ☞ equality operator ===를 제외한 comparison operator의 operand에 undefined나 null이 오지 않도록 특별히 주의한다.
- ☞ 또한, undefined나 null이 될 가능성이 있는 변수가 <, >, <=, >=의 operand가 되지 않도록 주의한다. 명확한 의도를 갖고 있지 않은 이상 주의한다. 만약 변수가 undefined나 null이 될 가능성이 있다고 판단되면, 이를 따로 처리하는 코드를 추가하기를 권장한다.

요약하면,

- ☞ comparison operator는 boolean value를 반환한다.
- ☞ string은 문자 단위로 비교되는데, 이때 비교 기준은 'dictionary' 순이다.
- ☞ 서로 다른 type의 값을 비교할 땐 number로 형 변환이 이뤄지고 난 후 비교가 진행된다(equality operator는 제외).
- ☞ null과 undefined는 euals comparison(동등 비교)(==) 시 서로 같지만 다른 값과는 같지 않다.
- ☞ null이나 undefined가 될 확률이 있는 변수가 > 또는 < 의 operand로 올 때는 주의를 기울인다. null, undefined 여부를 확인하는 코드를 따로 추가하는 습관을 갖기를 권장한다.

**[ 과제 ]**

☞ Comparison

★ **중요도:** 5

아래 표현식들의 결과를 예측해보자.

```
5 > 4
"apple" > "pineapple"
"2" > "12"
undefined == null
undefined === null
null == "\n0\n"
null === +"\n0\n"
```

**[ 해답 ]**

```
5 > 4                    → true
"apple" > "pineapple"    → false
"2" > "12"               → true
undefined == null        → true
undefined === null       → false
null == "\n0\n"          → false
null === +"\n0\n"        → false
```

① 명백히 true 이다.
② string의 비교는 사전순서가 기준이므로 false 이다. "a"는 "p"보다 작다.
③ 두 operand는 string이므로, 사전순으로 비교가 이뤄진다. 왼쪽 operand의 첫 번째 글자 "2"는 오른쪽 operand의 첫 번째 글자 "1"보다 크다.
④ null과 undefined는 같다.
⑤ equality operator는 type도 체크한다. 형이 다르면 false가 반환된다.
⑥ (4)와 유사한 문제이다. null은 오직 undefined와 같다.
⑦ type이 다르므로 false가 반환된다.