Logical Operator (logical 연산자)

☑ Logical Operator

JavaScript엔 세 종류의 logical operator ||(OR), &&(AND), !(NOT)이 있다.

operator에 'logical' 이라는 수식어가 붙긴 하지만 logical operator는 operand로 boolean 뿐만 아니라 모든 type의 값을 받을 수 있다. 연산 결과 역시 모든 type이 될 수 있다.

좀 더 자세히 알아보도록 한다.

❖ || (OR)

'OR' operator는 두 개의 수직선 기호로 만들 수 있다.

```
result = a || b;
```

전통적인 프로그래밍에서 OR operator는 boolean value를 조작하는 데 쓰인다. argument 중 하나라도 true이면 true를 반환하고, 그렇지 않으면 false를 반환한다.

JavaScript의 OR operator는 다루긴 까다롭지만 강력한 기능을 제공한다. OR을 어떻게 응용할 수 있는지 알아보기 전에 먼저, OR operator가 boolean value을 어떻게 다루는지 알아보도록 한다.

OR operator는 binary operator이므로 아래와 같이 4가지 조합이 가능하다.

```
alert( true || true );   // true
alert( false || true );  // true
alert( true || false );  // true
alert( false || false ); // false
```

operand(연산에 참여하는 변수나 상수)가 모두 false인 경우를 제외하고 연산 결과는 항상 true 이다. operand가 boolean이 아니면, 평가를 위해 boolean으로 변환된다.

예를 들어, 연산 과정에서 숫자 1은 true로, 숫자 0은 false로 바뀐다.

```
if (1 || 0) {            // if( true || false ) 와 동일하게 동작한다.
  alert( 'truthy!' );
}
```

OR operator ||은 if 문에서 자주 사용된다. 주어진 조건 중 하나라도 true인지를 테스트하는 용도로 사용된다.

[예시]

```
let hour = 9;

if (hour < 10 || hour > 18) {
  alert( '영업시간이 아닙니다.' );
}
```

if 문 안에 여러 가지 조건을 넣을 수 있다.

```
let hour = 12;
let isWeekend = true;

if (hour < 10 || hour > 18 || isWeekend) {
  alert( '영업시간이 아닙니다.' ); // 주말이기 때문임
}
```

❖ 첫 번째 truthy를 찾는 OR operator '||'

지금까진 operand가 boolean인 경우만을 다뤘다. 전통적인 방식이다. 이제 JavaScript에서만 제공하는 logical operator OR의 '추가'기능에 대해 알아보자.

추가 기능은 아래와 같은 algorithm으로 동작합니다.

OR operator와 operand가 여러 개인 경우:

```
result = value1 || value2 || value3;
```

이때, OR ||operator는 다음 순서에 따라 연산을 수행한다.

① 가장 왼쪽 operand부터 시작해 오른쪽으로 나아가며 operand를 평가한다.
② 각 operand를 boolean으로 변환한다. 변환 후 그 값이 true이면 연산을 멈추고, 해당 operand의 변환 전 original value(원래 값)을 반환한다.
③ operand 모두를 평가한 경우(모든 operand가 false로 평가되는 경우)엔 마지막 operand를 반환한다.

여기서 핵심은 반환 값이 형 변환을 하지 않은 원래 값이라는 것이다.

정리해 보면, OR "||" operator를 여러 개 체이닝(chaining) 하면 첫 번째 truthy를 반환한다. operand에 truthy가 하나도 없다면 마지막 operand를 반환한다.

[예시]

```
alert( 1 || 0 );                 // 1 (1은 truthy임)

alert( null || 1 );              // 1 (1은 truthy임)
alert( null || 0 || 1 );         // 1 (1은 truthy임)

alert( undefined || null || 0 );  // 0 (모두 falsy이므로, 마지막 값을 반환함)
```

이런 OR operator의 추가 기능을 이용하면 여러 용도로 OR operator를 활용할 수 있다.

☞ 변수 또는 표현식으로 구성된 목록에서 첫 번째 truthy 얻기

firstName, lastName, nickName이란 변수가 있는데 이 값들은 모두 옵션 값이라고 해보자.
OR ||을 사용하면 실제 값이 들어있는 변수를 찾고, 그 값을 보여줄 수 있다. 변수 모두에 값이 없는 경우엔 "Smart"를 보여준다.

```
let firstName = "";
let lastName = "";
let nickName = "Smart";

alert( firstName || lastName || nickName || "Smart");       // Smart
```

모든 변수가 falsy이면 "Smart"가 출력되었을 것이다.

☞ Short-circuit evaluation(단락 평가)

OR operator ||가 제공하는 또 다른 기능은 '단락 평가(short circuit evaluation)' 이다.

위에서 설명한 것과 같이 OR||은 왼쪽부터 시작해서 오른쪽으로 평가를 진행하는데, truthy를 만나면 나머지 값들은 건드리지 않은 채 평가를 멈춘다. 이런 프로세스를 'short circuit evaluation(단락 평가)'라고 한다.

단락 평가의 동작 방식은 두 번째 operand가 변수 할당과 같은 부수적인 효과(side effect)를 가지는 표현식 일 때 명확히 볼 수 있다.

아래 예시를 실행하면 두 번째 메시지만 출력된다.

```
true || alert("not printed");
false || alert("printed");
```

첫 번째 줄의 || operator는 true를 만나자마자 평가를 멈추기 때문에 alert가 실행되지 않는다.
단락 평가는 operator 왼쪽 조건이 falsy일 때만 명령어를 실행하고자 할 때 자주 쓰인다.

❖ && (AND)

두 개의 ampersand를 연달아 쓰면 AND operator &&를 만들 수 있다.

```
result = a && b;
```

전통적인 프로그래밍에서 AND operator는 두 operand가 모두가 참일 때 true를 반환한다. 그 외의 경우는 false를 반환한다.

```
alert( true && true );   // true
alert( false && true );  // false
alert( true && false );  // false
alert( false && false ); // false
```

아래는 if 문과 AND operator를 함께 활용한 예제이다.

```
let hour = 12;
let minute = 30;

if (hour == 12 && minute == 30) {
  alert( '현재 시각은 12시 30분입니다.' );
}
```

OR operator와 마찬가지로 AND operator의 operand도 type에 제약이 없다.

```
if (1 && 0) { // operand가 숫자형이지만 logical형으로 바뀌어 true && false가된다.
  alert( "if 문 안에 falsy가 들어가 있으므로 alert 창은 실행되지 않습니다." );
}
```

☞ 첫 번째 falsy를 찾는 AND operator '&&'

AND operator와 operand가 여러 개인 경우를 살펴보자.

```
result = value1 && value2 && value3;
```

AND operator &&는 아래와 같은 순서로 동작한다.

① 가장 왼쪽 operand부터 시작해 오른쪽으로 나아가며 operand를 평가한다.
② 각 operand는 boolean으로 변환된다. 변환 후 값이 false이면 평가를 멈추고, 해당 operand의 변환 전 원래 값을 반환한다.
③ operand 모두가 평가되는 경우(모든 operand가 true로 평가되는 경우)엔 마지막 operand가 반환된다.

정리해 보면, AND operator는 첫 번째 falsy를 반환한다. operand에 falsy가 없다면 마지막 값을 반환한다.

위 algorithm은 OR operator의 algorithm과 유사하다. 차이점은 AND operator가 첫 번째 falsy를 반환하는 반면, OR은 첫 번째 truthy를 반환한다는 것이다.

[예시]

```
// 첫 번째 operand가 truthy이면,
// AND는 두 번째 operand를 반환한다.
alert( 1 && 0 ); // 0
alert( 1 && 5 ); // 5

// 첫 번째 operand가 falsy이면,
// AND는 첫 번째 operand를 반환하고, 두 번째 operand는 무시한다.
alert( null && 5 );       // null
alert( 0 && "아무거나 와도 상관없습니다." ); // 0
```

AND operator에도 operand 여러 개를 연속해서 전달할 수 있다. 첫 번째 falsy가 어떻게 반환되는지 예시를 통해 살펴보자.

```
alert( 1 && 2 && null && 3 ); // null
```

아래 예시에선 AND operator의 operand가 모두 truthy이기 때문에 마지막 operand가 반환된다.

```
alert( 1 && 2 && 3 ); // 마지막 값, 3
```

⊙ &&의 우선순위가 ||보다 높다.

AND operator &&의 우선순위는 OR operator || 보다 높다.

따라서 a && b || c && d는 (a && b) || (c && d)와 동일하게 동작한다.

if를 ||나 &&로 대체하여 사용하지 않도록 한다.
어떤 개발자들은 AND operator &&를 if 문을 '짧게' 줄이는 용도로 사용하곤 한다.

[예시]

```
let x = 1;

(x > 0) && alert( '0보다 큽니다!' );
```

&&의 오른쪽 operand는 평가가 && 우측까지 진행되어야 실행된다. 즉, (x > 0)이 참인 경우에만 alert 문이 실행된다.

위 코드를 if 문을 써서 바꾸면 다음과 같다.

```
let x = 1;

if (x > 0) alert( '0보다 큽니다!' );
```

&&를 사용한 코드가 더 짧긴 하지만 if 문을 사용한 예시가 코드에서 무엇을 구현하고자 하는지 더 명백히 드러내고, 가독성도 좋다. 따라서 if 조건문이 필요하면 if를 사용하고, AND operator는 operator 목적에 맞게 사용한다.

❖ ! (NOT)

logical operator NOT은 느낌표 !를 써서 만들 수 있다.
NOT operator의 문법은 매우 간단하다.

```
result = !value;
```

NOT operator는 인수를 하나만 받고, 다음 순서대로 연산을 수행한다.
① operand를 boolean(true / false)으로 변환한다.
② 1에서 변환된 값의 역을 반환한다.

[예시]

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

NOT을 두 개 연달아 사용(!!)하면 값을 boolean으로 변환할 수 있다.

```
alert( !!"non-empty string" );    // true
alert( !!null );                  // false
```

이때, 첫 번째 NOT operator는 operand로 받은 값을 boolean으로 변환한 후 이 값의 역을 반환하고, 두 번째 NOT operator는 첫 번째 NOT operator가 반환한 값의 역을 반환한다. 이렇게 NOT을 연달아 사용하면 특정 값을 boolean으로 변환할 수 있다.

참고로, 내장 함수 Boolean을 사용하면 !!을 사용한 것과 같은 결과를 도출할 수 있다.

```
alert( Boolean("non-empty string") );      // true
alert( Boolean(null) );                    // false
```

NOT operator의 우선순위는 모든 logical operator 중에서 가장 높기 때문에 항상 &&나 || 보다 먼저 실행된다.

[과제]

☞ 다음 OR 연산의 결과는 무엇일까요?

★ 중요도: 5

아래 코드의 결과를 예측해 보자.

```
alert( null || 2 || undefined );
```

[해답]

operand 중 첫 번째 truthy인 2가 출력된다.

```
alert( null || 2 || undefined );
```

☞ OR operator의 operand가 alert 라면?

★ 중요도: 3

아래 코드의 결과를 예측해 보자.

```
alert( alert(1) || 2 || alert(3) );
```

[해답]

alert 창엔 1, 2가 차례대로 출력된다.

```
alert( alert(1) || 2 || alert(3) );
```

alert method는 값을 반환하지 않는다. 즉, undefined를 반환한다.

① 첫 번째 OR || 은 왼쪽 operand인 alert(1)를 평가한다. 이때 첫 번째 얼럿 창에 1이 출력된다.
② alert method는 undefined를 반환하기 때문에, OR operator는 다음 operand를 평가하게 된다. truthy를 찾기 위해서 이다.
③ 두 번째 operand(오른쪽 operand)인 2는 truthy이기 때문에 실행이 멈추고 2가 반환된다. 반환된 값 2는 제일 바깥 alert의 operand가 되어 두 번째 alert 창에 출력된다.

평가가 alert(3)까지 진행되지 않기 때문에 3은 출력되지 않는다.

☞ 다음 AND 연산의 결과는 무엇일까?

★ 중요도: 5

아래 코드의 결과를 예측해 보자.

```
alert( 1 && null && 2 );
```

[해답]

operand 중 첫 번째 falsy인 null이 출력된다.

```
alert( 1 && null && 2 );
```

☞ AND operator의 operand가 alert 라면?

★ **중요도:** 3

아래 코드의 결과를 예측해 보자.

```
alert( alert(1) && alert(2) );
```

[해답]

alert 창엔 1, undefined가 차례대로 출력된다.

```
alert( alert(1) && alert(2) );
```

alert를 호출하면 undefined가 반환된다. alert는 단순히 얼럿 창에 메시지만 띄워주고, 의미 있는 값을 반환해 주지 않는다.

&&는 왼쪽 operand를 평가하고(이때 1이 alert 창에 출력된다) 평가를 즉시 멈춘다. alert(1)의 평가 결과는 undefined로 falsy이기 때문이다. && operator는 falsy를 만나면 그 값을 출력하고 즉시 연산을 멈춘다.

☞ OR AND OR operator로 구성된 표현식

★ **중요도:** 5

아래 코드의 결과를 예측해 보자.

```
alert( null || 2 && 3 || 4 );
```

[해답]

alert 창엔 3이 출력된다.

```
alert( null || 2 && 3 || 4 );
```

AND operator &&의 우선순위는 ||보다 높다. 따라서 &&가 먼저 실행된다.

2 && 3 = 3이므로, 문제에서 제시한 표현식은 아래와 같이 바꿔쓸 수 있다.

```
null || 3 || 4
```

따라서 첫 번째 truthy인 3이 출력된다.

☞ 사이 범위 확인하기

★ **중요도:** 3

age(나이)가 14세 이상 90세 이하에 속하는지를 확인하는 if문을 작성하자.
"이상과 이하"는 age(나이) 범위에 14나 90이 포함된다는 의미이다.

[해답]

```
if (age >= 14 && age <= 90)
```

☞ 바깥 범위 확인하기

★ **중요도:** 3

age(나이)가 14세 이상 90세 이하에 속하지 않는지를 확인하는 if문을 작성하자.
답안은 NOT ! operator를 사용한 답안과 사용하지 않은 답안 2가지를 제출하도록 한다.

[해답]

첫 번째 답안은 다음과 같다.

```
if (!(age >= 14 && age <= 90))
```

두 번째 답안은 다음과 같다.

```
if (age < 14 || age > 90)
```

☞ "if"에 관한 고찰

★ **중요도:** 5

아래 표현식에서 어떤 alert가 실행될까?
if(...) 안에 표현식이 있으면 어떤 일이 일어날까?

```
if (-1 || 0) alert( 'first' );
if (-1 && 0) alert( 'second' );
if (null || -1 && 1) alert( 'third' );
```

[해답]

첫 번째 표현식과 세 번째 표현식에 있는 alert가 실행된다.

이유:

```
// -1 || 0 은 -1 이므로 truthy 이다.
// 따라서 alert가 실행된다.
```

```
if (-1 || 0) alert( 'first' );

// -1 && 0 은 0 이므로 falsy 이다.
// 따라서 alert가 실행되지 않는다.
if (-1 && 0) alert( 'second' );

// operator &&는 ||보다 우선순위가 높다.
// 따라서 -1 && 1 이 먼저 실행되어 아래와 같이 표현식이 순차적으로 바뀐다.
// null || -1 && 1  ->  null || 1  ->  1
// 결과적으로 alert가 실행된다.
if (null || -1 && 1) alert( 'third' );
```

☞ 로그인 구현하기

★ **중요도:** 3

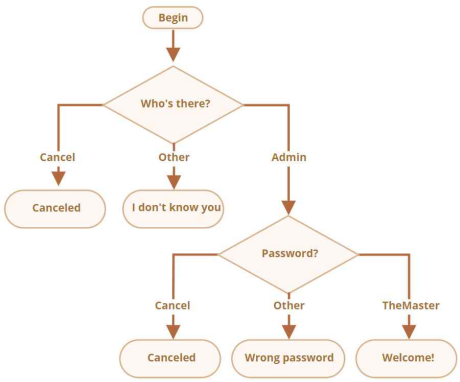
프롬프트(prompt) 대화상자를 이용해 간이 로그인 창을 구현해보자.

사용자가 "Admin"를 입력하면 비밀번호를 물어보는 prompt 대화상자를 띄워준다. 이때 아무런 입력도 하지 않거나 Esc를 누르면 "취소되었습니다."라는 메시지를 보여준다. 틀린 비밀번호를 입력했다면 "인증에 실패하였습니다."라는 메시지를 보여준다.

비밀번호 확인 절차는 다음과 같다.

① 맞는 비밀번호 "TheMaster"를 입력했다면 "환영합니다!"라는 메시지를 보여준다.
② 틀린 비밀번호를 입력했다면 "인증에 실패하였습니다."라는 메시지를 보여준다.
③ 빈 문자열을 입력하거나 입력을 취소했다면 "취소되었습니다."라는 메시지를 보여준다.

순서도는 다음과 같다.



중첩 if 블록을 사용하고, 코드 전체의 가독성을 고려해 답안을 작성하시오.

힌트: prompt 창에 아무것도 입력하지 않으면 빈 string인 ''가, ESC를 누르면 null이 반환된다.

[해답]

```
let userName = prompt("사용자 이름을 입력해주세요.", '');

if (userName == 'Admin') {
  let pass = prompt('비밀번호:', '');
  if (pass == 'TheMaster') {
    alert( '환영합니다!' );
  } else if (pass == '' || pass == null) {
    alert( '취소되었습니다.' );
  } else {
    alert( '인증에 실패하였습니다.' );
  }
} else if (userName == '' || userName == null) {
  alert( '취소되었습니다.' );
} else {
  alert( "인증되지 않은 사용자입니다." );
}
```

if 블록 안쪽의 들여쓰기를 주의 깊게 보아야 한다. 들여쓰기는 필수가 아니지만, 코드 가독성을 높이는 데 도움을 준다.