8주차 JavvaScript

상태	승인
설명	논리연산자, 반복문,switch

논리 연산자

자바스크립트엔 세 종류의 논리 연산자 `||`(OR), `&&`(AND), `!`(NOT)이 존재한다.

|| (OR)

'OR' 연산자는 두 개의 수직선 기호로 만들 수 있다.

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

`a`, `b` 중 하나라도 `true`이면 `true`를 반환하고, 그렇지 않으면 `false`를 반환하게 된다.

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

피연산자가 모두 `false`인 경우를 제외하고 연산 결과는 항상 `true` 가 나오게 된다.

`OR`을 한국어로 하면 `또는` 이란 의미를 가지는데 해당의미와 동일하게 연산되는 값중 하나라도 `true` 가 나오면 `true`를 반환하게 된다.

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

OR 연산자 `||`은 `if`문에서 자주 사용된다.

```
let hour = 9;

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

`||`을 활용하면 `if`문 안에 여러 가지 조건을 넣는것이 가능하다.

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

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

&& (AND)

두 개의 앰퍼샌드를 연달아 쓰면 AND 연산자 `&&`를 만들 수 있다.

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

AND 연산자는 두 피연산자가 모두가 참일 때 `true`를 반환하며 그 외의 경우는 `false`를 반환한다.

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

AND 역시 if문과 함께 자주 사용된다.

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

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

`OR` 연산자와 마찬가지로 `AND` 연산자의 피연산자도 타입에 제약이 없다. 즉, 어떤 값을 넣어도 `true` 혹은 `false`로 변환되어 연산이 진행된다.

```
if (1 && 0) { // 피연산자가 숫자형이지만 논리형으로 바뀌어 true && false가 됩니다.
  alert( "if 문 안에 falsy가 들어가 있으므로 alert창은 실행되지 않습니다." );
}
```

! NOT

논리 연산자 NOT은 느낌표 `!`를 써서 만들 수 있으며 문법이 매우 간결하다.

```
result = !value;
```

NOT 연산자는 아래와 같은 동작을 한다.

1️⃣ 피연산자를 불린형(`true / false`)으로 변환합니다.

2️⃣ 1에서 변환된 값의 역을 반환합니다.

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

이러한 `NOT`의 연산을 활용하면 `boolean` 형변환을 조금 더 쉽게 할 수 있다.

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

즉, 아래 코드와 위 코드는 같은 동작을 하게 된다.

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

`NOT` 연산자의 우선순위는 모든 논리 연산자 중에서 가장 높기 때문에 항상 `&&`나 `||` 보다 먼저 실행된다!

문제 1

아래 코드의 결과를 예측해 보세요.

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

문제 2

아래 코드의 결과를 예측해 보세요.

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

문제 3

아래 코드의 결과를 예측해 보세요.

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

문제 4

`age`(나이)가 `14`세 이상 `90`세 이하에 속하는지를 확인하는 `if`문을 작성하세요.

"이상과 이하"는 `age`(나이) 범위에 `14`나 `90`이 포함된다는 의미입니다.

문제 5

`age`(나이)가 `14`세 이상 `90`세 이하에 속하지 않는지를 확인하는 `if`문을 작성하세요.

답안은 NOT `!` 연산자를 사용한 답안과 사용하지 않은 답안 2가지를 작성하세요

while과 for 반복문

개발을 하다 보면 여러 동작을 반복해야 하는 경우가 종종 생긴다.

상품 목록에서 상품을 차례대로 출력하거나 숫자를 1부터 10까지 하나씩 증가시키면서 동일한 코드를 반복 하는 경우처럼 말이다.

`반복문(loop)` 을 사용하면 동일한 코드를 여러 번 반복할 수 있으며 방금 설명한 상황에서 매우 효율적으로 사용할 수 있다.

while

`while` 반복문의 문법은 아래와 같다.

```
while (condition) {
  // 코드
  // '반복문 본문(body)'이라 불림
}
```

`condition`(조건)이 truthy 이면 반복문 본문의 `코드`가 실행되게 된다. 이전에 배웠던 `if`문과 상당히 유사하단 걸 알 수 있다.

아래 반복문은 조건 `i < 3`을 만족할 동안 `i`를 출력해준다.

```
let i = 0;
while (i < 3) { // 0, 1, 2가 출력됩니다.
  alert( i );
  i++;
}
```

반복문 조건엔 비교뿐만 아니라 모든 종류의 표현식, 변수가 올 수 있다. 이 역시 `if`문과 동일하다.

아래 예시에선 `while (i != 0)`을 짧게 줄여 `while (i)`로 만들어 본 예제이다.

```
let i = 3;
while (i) { // i가 0이 되면 조건이 falsy가 되므로 반복문이 멈춥니다.alert( i );
  i--;
}
```

본문이 한 줄이면 대괄호를 쓰지 않아도 된다.

반복문 본문이 한 줄짜리 문이라면 대괄호 `{…}`를 생략할 수 있다. 역시 `if`문과 동일하다.

```
let i = 3;
while (i) alert(i--);
```

do…while

`do..while` 문법을 사용하면 `condition`을 반복문 본문 *아래*로 옮길 수 있다.

```
do {
  // 반복문 본문
} while (condition);
```

이때 본문이 먼저 실행되고, 조건을 확인한 후 조건이 truthy인 동안엔 본문이 계속 실행되게 된다.

```
let i = 0;
do {
  alert( i );
  i++;
} while (i < 3);
```

`do..while` 문법은 조건이 truthy 인지 아닌지에 상관없이, 본문을 **최소한 한 번**이라도 실행하고 싶을 때사용할 수 있는 문법이다. 대다수의 상황에선 `do..while`보다 `while(…) {…}`이 더 많이 사용된다.

for

`for` 반복문은 `while` 반복문보다는 복잡하지만 가장 많이 쓰이는 반복문이며 문법은 아래와 같다.

```
for (begin; condition; step) {
  // ... 반복문 본문 ...
}
```

`for`문을 구성하는 각 요소가 무엇을 의미하는지 알아보자. 아래 반복문을 실행하면 `i`가 `0`부터 `3`이 될 때까지(단, `3`은 포함하지 않음) `alert(i)`가 호출되게 된다.

```
for (let i = 0; i < 3; i++) { // 0, 1, 2가 출력됩니다.
  alert(i);
}
```

이제 `for`문의 구성 요소를 하나씩 살펴보자.

구성 요소		
begin	`i = 0`	반복문에 진입할 때 단 한 번 실행됩니다.
condition	`i < 3`	반복마다 해당 조건이 확인됩니다. false이면 반복문을 멈춥니다.
body	`alert(i)`	condition이 truthy일 동안 계속해서 실행됩니다.
step	`i++`	각 반복의 body가 실행된 이후에 실행됩니다.

즉, 일반적으로 반복문은 아래와 같은 방식으로 실행되게 된다.

```
begin을 실행함
→ (condition이 truthy이면 → body를 실행한 후, step을 실행함)
→ (condition이 truthy이면 → body를 실행한 후, step을 실행함)
→ (condition이 truthy이면 → body를 실행한 후, step을 실행함)
→ ...
```

`begin`이 한 차례 실행된 이후에, `condition` 확인과 `body`, `step`이 계속해서 반복 실행되는 것이다.

이해가 잘 되지 않는다면 아래 코드를 천천히 따라가면서 다시한번 이해해보자.

첫번째 주석처리 된 for문과 그 아래 코드는 같은 동작을 하게 된다.

```
// for (let i = 0; i < 3; i++) alert(i)

// begin을 실행함
let i = 0
// condition이 truthy이면 → body를 실행한 후, step을 실행함
if (i < 3) { alert(i); i++ }
// condition이 truthy이면 → body를 실행한 후, step을 실행함
if (i < 3) { alert(i); i++ }
// condition이 truthy이면 → body를 실행한 후, step을 실행함
if (i < 3) { alert(i); i++ }
// i == 3이므로 반복문 종료
```

인라인 변수 선언

지금까진 '카운터' 변수 `i`를 반복문 안에서 선언하였다. 이런 방식을 `인라인 변수` 선언이라고 부르며 이렇게 선언한 변수는 반복문 안에서만 접근할 수 있다.

```
for (let i = 0; i < 3; i++) {
  alert(i); // 0, 1, 2
}
alert(i); // Error: i is not defined
```

반복문 안❓

`for`, `while`, `if` 등 여러 문은 본인의 영역이 존재한다. `if`문을 예로들면 `()` 안의 값이 `true`일때 `{}` 안의 영역이 실행되게 하는데, 이때 `{}` 안을 `if`문 영역이라고 칭한다.

즉, `for`문 에서 `()`안에서 `i`를 `let`으로 선언하였기 때문에 `for`문의 `{}` 안에서만 `i` 는 살아있게 되는 것이다.

`인라인 변수` 선언 대신, 정의되어 있는 변수를 사용할 수도 있다.

```
let i = 0;

for (i = 0; i < 3; i++) { // 기존에 정의된 변수 사용
  alert(i); // 0, 1, 2
}

alert(i); // 3, 반복문 밖에서 선언한 변수이므로 사용할 수 있음
```

반복문 빠져나오기

대개는 반복문의 조건이 falsy가 되면 반복문이 종료된다.

그런데 특별한 지시자인 `break`를 사용하면 언제든 원하는 때에 반복문을 빠져나올 수 있다.

아래 예시의 반복문은 사용자에게 일련의 숫자를 입력하도록 안내하고, 사용자가 아무런 값도 입력하지 않으면 반복문을 '종료'하는 예제이다.

```
let sum = 0;

while (true) {

  let value = +prompt("숫자를 입력하세요.", '');

  if (!value) break; // (*)

  sum += value;

}
alert( '합계: ' + sum );
```

`(*)`로 표시한 줄에 있는 `break`는 사용자가 아무것도 입력하지 않거나 `Cancel`버튼을 눌렀을 때 활성화된다. 이때 반복문이 즉시 중단되고 `while`문 밖으로 나가게 된다.

반복문의 시작 지점이나 끝 지점에서 조건을 확인하는 것이 아니라 본문 가운데 혹은 본문 여러 곳에서 조건을 확인해야 하는 경우, '무한 반복문 + `break`' 조합을 사용하면 좋을 것이다.

다음 반복으로 넘어가기

`continue` 지시자는 `break`의 '가벼운 버전'이라 생각할 수 있다.

`continue`는 전체 반복문을 멈추지 않으며 대신에 현재 실행 중인 반복을 멈추고 다음 반복으로 넘어가게 된다.

아래 예시를 보면서 이해해보자.

아래 반복문은 `continue`를 사용해 홀수를 출력하는 예제이다.

```
for (let i = 0; i < 10; i++) {

  // 조건이 참이라면 남아있는 본문은 실행되지 않습니다.
  if (i % 2 == 0) continue;alert(i); // 1, 3, 5, 7, 9가 차례대로 출력됨
}
```

요약

- `while` – 각 반복이 시작하기 전에 조건을 확인
- `do..while` – 각 반복이 끝난 후에 조건을 확인
- `for (;;)` – 각 반복이 시작하기 전에 조건을 확인하며 추가 세팅이 가능하다.

'무한' 반복문은 보통 `while(true)`를 써서 만들며 무한 반복문은 여타 반복문과 마찬가지로 `break` 지시자를 사용해 멈출 수 있다.

현재 실행 중인 반복에서 더는 무언가를 하지 않고 다음 반복으로 넘어가고 싶다면 `continue` 지시자를 사용하면 된다.

문제 1

아래 코드를 실행했을 때 얼럿 창에 마지막으로 뜨는 값은 무엇일까요? 이유도 함께 설명해보세요.

```
let i = 3;

while (i) {
  alert( i-- );
}
```

문제 2

while 반복문이 순차적으로 실행될 때마다 얼럿 창에 어떤 값이 출력될지 예상해 보세요.

아래 두 예시는 같은 값을 출력할까요?

1. 전위형 증가 연산자를 사용한 경우(`++i`):

```
let i = 0;
while (++i < 5) alert( i );
```

2. 후위형 증가 연산자를 사용한 경우(`i++`):

```
let i = 0;
while (i++ < 5) alert( i );
```

문제 3

`for` 반복문을 이용하여 `2`부터 `10`까지 숫자 중 짝수만을 출력해보세요.

문제 4

`for` 반복문을 `while` 반복문으로 바꾸되, 동작 방식에는 변화가 없도록 해보세요. 출력 결과도 동일해야 합니다.

```
for (let i = 0; i < 3; i++) {
  alert( `number ${i}!` );
}
```

문제 5

사용자가 `100`보다 큰 숫자를 입력하도록 안내하는 프롬프트 창을 띄워보세요. 사용자가 조건에 맞지 않은 값을 입력한 경우 반복문을 사용해 동일한 프롬프트 창을 띄워줍시다.

사용자가 `100`을 초과하는 숫자를 입력하거나 취소 버튼을 누른 경우, 혹은 아무것도 입력하지 않고 확인 버튼을 누른 경우엔 더는 프롬프트 창을 띄워주지 않아도 됩니다.

사용자가 오직 숫자만 입력한다고 가정하고 답안을 작성하도록 해봅시다. 숫자가 아닌 값이 입력되는 예외 상황은 처리하지 않아도 됩니다.

💡 문제 6

소수는 자신보다 작은 두 개의 자연수를 곱하여 만들 수 없는 1보다 큰 자연수입니다.

다시 말해서 `1`과 `그 수 자신` 이외의 자연수로는 나눌 수 없는 자연수를 소수라고 부르죠.

`5`는 `2`나 `3`, `4`로 나눌 수 없기 때문에 소수입니다. `5`를 이들 숫자로 나누면 나머지가 있기 때문이죠.

2부터 `n`까지의 숫자 중 소수만 출력해주는 코드를 작성해봅시다.

`n = 10`이라면 결과는 `2,3,5,7`이 되어야겠죠.

주의: 작성한 코드는 임의의 숫자 `n`에 대해 동작해야 합니다.

switch문

복수의 `if` 조건문은 `switch`문으로 바꿀 수 있다.

`switch`문을 사용한 비교법은 특정 변수를 다양한 상황에서 비교할 수 있게 해준다.

문법

`switch`문은 하나 이상의 `case`문으로 구성된다.

```
switch(x) {
  case 'value1':  // if (x === 'value1')
    ...
    [break]

  case 'value2':  // if (x === 'value2')
    ...
    [break]

  default:
    ...
    [break]
}
```

1️⃣ 변수 `x`의 값과 첫 번째 `case`문의 값 `'value1'`를 일치 비교한 후, 두 번째 `case`문의 값 `'value2'`와 비교하고 이 과정은 계속 이어서 진행한다.

2️⃣ `case`문에서 변수 `x`의 값과 일치하는 값을 찾으면 해당 `case` 문의 아래의 코드가 실행된다. 이때, `break`문을 만나거나 `switch` 문이 끝나면 코드의 실행은 멈춘다.

3️⃣ 값과 일치하는 `case`문이 없다면, `default`문 아래의 코드가 실행된다.(`default` 문이 있는 경우).

switch문 예제를 한번 실습해보면서 과정을 이해해보자.

```
let a = 2 + 2;

switch (a) {
  case 3:
    alert( '비교하려는 값보다 작습니다.' );
    break;
  case 4:
    alert( '비교하려는 값과 일치합니다.' );
    break;case 5:
    alert( '비교하려는 값보다 큽니다.' );
    break;
  default:
    alert( "어떤 값인지 파악이 되지 않습니다." );
}
```

`switch`문은 `a`의 값인 `4`와 첫 번째 `case`문의 값인 `3`을 비교하고 두 값은 같지 않기 때문에 다음 `case`문으로 넘어가게 된다.

`a`와 그다음 `case`문의 값인 `4`는 일치한다. 따라서 `break`문을 만날 때까지 `case 4` 아래의 코드가 실행되게 된다.

`case`문 안에 `break`문이 없으면 조건에 부합하는지 여부를 따지지 않고 이어지는 `case`문을 실행한다.

해당 부분이 약간 헷갈릴 수 있으니 아래 예시를 보면서 동작하는 원리를 확인해보자.

```
let a = 2 + 2;
```

```
switch (a) {
  case 3:
    alert( '비교하려는 값보다 작습니다.' );
  case 4:
    alert( '비교하려는 값과 일치합니다.' );
  case 5:
    alert( '비교하려는 값보다 큽니다.' );
  default:
    alert( "어떤 값인지 파악이 되지 않습니다." );}
```

위 예시를 실행하면 아래 3개의 `alert`문이 실행되게 된다. (한번 실습해보자)

```
alert( '비교하려는 값과 일치합니다.' );
alert( '비교하려는 값보다 큽니다.' );
alert( "어떤 값인지 파악이 되지 않습니다." );
```

즉, `break`가 없다면 일치하는 `case():`의 코드부터 `default`까지 아래에 있는 모든 코드가 동작하게 된다.

여러개의 switch문 묶기

코드가 같은 `case`문은 위에서 설명했던 break의 성질을 통해서 묶을 수 있다.

`case 3`과 `case 5`에서 실행하려는 코드가 같은 경우에 대한 예시를 한번 알아보자.

```
let a = 3;

switch (a) {
  case 4:
    alert('계산이 맞습니다!');
    break;

  case 3: // (*) 두 case문을 묶음
  case 5:
    alert('계산이 틀립니다!');
    alert("수학 수업을 다시 들어보는걸 권유 드립니다.");
    break;
  default:
    alert('계산 결과가 이상하네요.');
}
```

`case 3` 에 아무코드가 없고, `break`문 또한 없기 때문에 `case5`의 코드가 실행되게 되며, `break`를 만나 `switch`문이 종료되게 된다.

즉 break속성을 잘 파악하면 위와같이 효율적으로 코드를 작성하는것이 가능해진다.

`if` vs `switch` ❓

사실 본질적으로 `if`문과 `switch` 문은 하는 역할이 동일하다. 모든 `if`문은 `switch`문으로, 모든 `switch`문은 `if`문으로 바꿀 수 있다.

그렇다면 어쩔때 `if`문과 `switch`문을 사용해야할까? 정해진 답은 없지만 보통 `if` - `else if` 문이 길어지고 안에 판별해야하는 변수가 1개인 경우에 `switch`문을 활용하면 가독성이 높아지게 된다.

따라서 적절한 상황에서 `if`문과 `switch`문을 활용해보자.

문제 1

"`switch`"문을 사용해 작성된 아래 코드를 `if..else`문을 사용한 코드로 변환해 보세요.

```
switch (browser) {
  case 'Edge':
    alert( "Edge를 사용하고 계시네요!" );
    break;

  case 'Chrome':
  case 'Firefox':
  case 'Safari':
  case 'Opera':
    alert( '저희 서비스가 지원하는 브라우저를 사용하고 계시네요.' );
    break;

  default:
    alert( '현재 페이지가 괜찮아 보이길 바랍니다!' );
}
```

문제 2

아래 코드를 `switch`문을 사용한 코드로 바꿔보세요. switch문은 하나만 사용해야 합니다.

```
let a = +prompt('a?', '');

if (a == 0) {
  alert( 0 );
}
if (a == 1) {
  alert( 1 );
}

if (a == 2 || a == 3) {
  alert( '2,3' );
}
```