Javascript 조건문

키워드 : 비교 연산 (https://www.w3schools.com/js/js_comparisons.asp),
논리 연산 (https://www.w3schools.com/js/js_booleans.asp),
if~else 조건문 (https://www.w3schools.com/js/js_if_else.asp),
switch~case 조건문 (https://www.w3schools.com/js/js_switch.asp),
break 구문 (https://www.w3schools.com/js/js_break.asp)

제가 강의 영상에서 프로그래밍은 논리적인 구조를 짜는 작업이라고 자주 언급했습니다. 논리적인 구조를 구성하기 위해 필수적으로 사용하게 되는 코드 구문이 이번 장에서 정리할 조건문입니다.

조건문은 특정 조건이 만족할 경우에 실행하고 싶은 코드를 하나의 블록({})으로 묶는 코드의 흐름 제어 구문입니다. 앞서 연산자에서 살펴본 비교 연산자와 이번 장에서 살펴볼 논리 연산자를 이용해 조건을 처리하게 됩니다.

비교 연산자에 대한 정리는 어느 정도 선행되었다고 생각하고 글을 작성하도록 하겠습니다.

논리 연산자

논리 연산자는 우리가 어떤 조건을 처리할 때 여러 조건을 동시에 비교하기 위해 사용하는 연산자입니다.

우선 논리 연산자의 종류는 다음과 같습니다.

연산자	의미	사용 범위
&&	논리곱(and)	모든 조건이 참일 경우
||	논리합(or)	조건 중 하나라도 참일 경우
!	부정(not)	값의 부정(반전)

아래 코드는 간단한 논리 연산자 사용의 예를 보여줍니다.

```
//1번
console.log(1 == 1 && 1 == 0); //false
//2번
console.log(1 == 1 || 1 == 0); //true
//3번
console.log(!true); //false
```

위 코드는 확인하고 싶은 조건을 논리 연산자로 구분하여 여러개의 조건을 비교하는 코드입니다. **and 연산**(&&)의 경우 모든 조건이 참일 경우 true를 반환합니다. 때문에 1번 코드의 경우 `1 == 0` 조건이 거짓이므로 false를 반환합니다. **or 연산**(||)의 경우 여러 조건 중 하나라도 참일 경우 false를 반환합니다. 때문에 2번 코드의 경우 `1 == 0` 조건은 거짓이지만 `1 == 1` 조건은 참이므로 true를 반환합니다. 마지막으로 **not 연산**(!)은 값의 부정을 나타냅니다. 3번 코드의 경우 true를 부정했기 때문에 값의 반전으로 false를 반환합니다.

참고로 소괄호()를 이용해 각 조건 별로 묶을 경우 코드의 가독성도 좋아지고, 조건 비교 실수를 방지할 수 있습니다.
e.g., (조건) && (조건)

또한, 논리 연산자는 제한 없이 여러 논리 조건을 동시에 사용할 수 있습니다. 예를 들면 아래와 같습니다.

```
var a = 10, b = 10, c = 10, d = 10;
console.log((a == b && b == c) || (a == c && c == d)); //true
```

참, 거짓을 나타내는 논리형(Boolean)은 true, false로 이루어져있습니다. 이렇게 자료형의 키워드인 true, false를 반환하는 데이터 타입이 있습니다.

> 값을 갖는 자료형 : true

> 값을 갖지 않는 자료형, 0 : false

아래 코드는 위 설명이 어떤 의미를 갖는지에 대한 간단한 예를 보여줍니다.

```
//1번
console.log(Boolean(100)); // true
console.log(Boolean(3.14)); //true
console.log(Boolean(-10)); //true
console.log(Boolean("hi")); //true
console.log(Boolean("false")); //true
console.log(Boolean(1 + 1 + "귀요미")); //true

//2번
console.log(Boolean(0)); //false
console.log(Boolean(-0)); //false
var str = "" //빈 문자열
console.log(Boolean(str)); //false
var x; //변수를 선언만 하고, 값을 할당하지 않은 경우
console.log(Boolean(x)); //false
var isNan = 10 / "A"; //NaN
console.log(Boolean(isNan)); //false
```

위에 코드에서 사용된 Boolean이란 함수는 아직 우리가 함수에 대한 개념을 다루지 않았기 때문에 완벽하게 이해하기 어렵지만, 값의 타입을 알려주는 typeof 연산자와 마찬가지로 Boolean(값) 형태로 작성하면 해당 값을 true, false로 반환해주는 Boolean 형태로 변환해준다고 이해해주시면 됩니다.

위 코드에서 1번 코드는 모두 **값이 있는** 형태입니다. 모두 true를 반환합니다. 2번 코드의 경우 0, -0와 **값이 없는** 형태입니다. 모두 false를 반환합니다.

여기서 **NaN**이란 키워드가 새롭게 등장한 것을 확인할 수 있습니다. NaN은 "Not a Number"의 약자입니다. 변수 "isNan"에 할당하려고 한 값은 10을 문자인 "A"로 나누려고 시도한 값입니다. 이처럼 숫자(Number)형이 아닌 자료형으로 결합을 위한 덧셈(+, +=) 연산을 제외한 산술 연산을 시도할 경우 NaN이란 값을 반환받게 됩니다.

if~else 구문

위에서 조건문은 특정 조건이 만족할 경우에 실행하고 싶은 코드를 하나의 블록({})으로 묶는 코드의 흐름 제어 구문이라고 설명했습니다.

우선, 흐름 제어를 위한 코드 블록 중 하나인 **if~else 구문**에 대해서 살펴보도록 하겠습니다.

`if~else` 구문은 다음과 같이 세 가지 구문으로 구성됩니다.

명칭	의미
if	조건이 true일 경우에 코드 블록({})의 실행 코드를 실행
else if	새로운 조건을 추가, 기존의 조건이 false일 경우 실행 코드를 실행
else	위에서 사용된 모든 조건이 false인 경우 실행 코드를 실행

`if~else` 구문은 아래와 같은 기본 문법 구조를 갖습니다.

```
//기본 형태
if (조건1) {
    //조건1이 true일 경우 실행하고 싶은 코드 작성.
}else if(조건2) {
    //조건1이 false, 조건2가 true일 경우 실행하고 싶은 코드 작성.
}else {
    //조건1, 조건 2가 모두 false일 경우 실행하고 싶은 코드 작성.
}

//코드 예시
var x = 10;
if (x < 100) {
    console.log("Good day"); //Good day
}
```

참고로, if~else 구문을 작성할 때 코드 블록({})을 작성하는 위치는 개발자의 마음입니다. 예를 들어, 다음과 같이 여러 가지 방법으로 코드 블록의 위치를 구성할 수 있습니다.

```
//1번
if(조건){

}else{

}
//2번
if(조건)
{

}
else
{

}
//3번
if(조건){}
else{}
//4번
//if~else 구문 다음에 오는 실행 코드가 한 줄(하나의 실행문)일 경우엔 코드 블록({})을 생략할 수 있습니다. 사실, 전 코드의 가독성을 위해 권장하지 않습니다. :)
if(조건)
    //실행코드
else
    //실행코드
```

switch~case 구문

다음은 흐름 제어를 위한 코드 블록 중 하나인 **switch~case 구문**에 대해서 살펴보도록 하겠습니다.

기본적으로 switch~case 구문도 if~else 구문과 마찬가지로 여러 조건에 따라 특정 동작(실행 코드)을 실행하고 싶을 때 사용하는 코드 블록({})입니다.

switch~case 구문은 아래와 같은 기본 문법 구조를 갖습니다.

```
switch(표현식) {
    case 값:
        //실행 코드
        break;
    case 값:
        //실행 코드
        break;
    default:
        //기본 실행 코드
}
```

다음은 switch~case 구문의 간단한 예시 코드입니다.

```
var x = 0;
switch(x) {
    case 0:
        console.log("0!"); //0!
        break;
    case 1:
        console.log("1!");
        break;
    default:
        console.log("not 0 and 1");
}
```

switch~case 구문은 위 예시처럼 조건이 아닌 표현식(값)에 따라서 코드의 실행을 분기(조건 처리)하고 싶을 경우에 사용할 수 있습니다.

마지막으로, 위에서 사용된 **break** 키워드에 대해서 간단하게 살펴보겠습니다.

break 키워드는 다음 장에서 살펴볼 반복문에서도 사용할 수 있는 키워드 중 하나입니다.

우리가 작성한 코드는 기본적으로 위에서부터 한 줄 한 줄 읽어 들여 실행되게 됩니다. switch~case 구문을 예로 들자면, 여러 조건에 따라 작성된 case 구문에서 만약 특정 case에 해당하는 표현식(값)을 찾았다면 해당 코드를 실행합니다. 이때 해당 case를 찾았으므로, 더 이상 코드를 실행할 필요 없이 switch~case 구문의 블록을 벗어나게 하는 것입니다.

또한, 위에 해당하는 case 값이 없을 경우 기본값으로 실행하고 싶은 default 키워드의 실행 코드는 마지막에 작성되어 있으므로, break 키워드를 작성하지 않아도 되는 것입니다. :)

위에서 살펴본 조건문은 Javascript 뿐만 아니라, 우리가 앞으로 공부할 Python, 그리고 Java, C, C++ 기타 등등 모든 프로그래밍 언어에서 사용되는 제어 구문입니다. 코드의 절반 가량은 이 조건문을 통해 구성된다고 해도 과언이 아닙니다.

논리 조건은 개발자가 직접 주는 것이기 때문에 그때그때 다른 방법으로 코드를 작성해야 하지만, 기본적인 조건문 사용법에 대해서는 꼭 이해하시길 바랍니다! :)