Javascript 연산자

키워드 : 자바스크립트 연산자 (https://www.w3schools.com/js/js_operators.asp)

우리가 수학 공부를 할 때 수식이라는 것을 작성하게 됩니다. 수식을 작성할 때 사용하는 여러 가지 기호가 존재합니다. e.g., 사칙연산(+, -, x, %) 프로그램 코드를 작성할 때도 이러한 기호가 사용되게 됩니다. 또한, 이것은 약속된 "규칙"을 나타낸다고 할 수 있습니다.

그렇다면, 우리가 어떻게 이 기호를 사용할 수 있는지 살펴보도록 하겠습니다. 우선 덧셈을 나타내는 수식은 아래와 같이 나타낼 수 있습니다.

```
1 + 1 = 귀요미?
```

여기서 "+" 기호는 덧셈을 나타냅니다. 이러한 기호를 우리는 연산자(Operators)라고 부릅니다. 이러한 연산자를 통해 우리는 어떤 변수에 값을 더하거나 뺄 수 있고, 여러 가지 논리적인 처리를 할 수 있습니다.

우선, 산술 연산자부터 살펴보도록 하겠습니다.

산술 연산자

산술 연산자는 이름에서 느껴지듯이 수학과 관련된 산술 연산을 할 수 있는 연산자입니다.

아래 코드는 x, y라는 변수를 선언하고, 각각 1을 할당한 후 덧셈 연산을 하는 코드입니다. 덧셈 연산을 실행하고 z라는 새로운 변수에 값을 할당하고,
"console.log"로 값을 확인하면, 결괏값이 2로 나타나는 것을 확인할 수 있습니다.

```
var x = 1, y = 1;
var z = x + y;
console.log(z); //2
```

위와 마찬가지로 사칙 연산을 할 수 있는 연산자는 다음과 같습니다.

연산자	의미	예
+	덧셈	var z = x + y;
-	뺄셈	var z = x - y;
*	곱셈	var z = x * y;
/	나눗셈	var z = x / y;
%	나머지	var z = x % y;
++	증가	var z = ++x;
--	감소	var z = --x;

위 산술 연산을 사용할 때, 전위 연산과 후위 연산에 대해 생각해볼 필요가 있습니다. 우리가 수학에서 사용하는 연산 방법은 중위 연산입니다. 연산자가 가운데 오고 피연산자가 앞 뒤로 오는 형태입니다. e.g., 1 + 1 = 2

프로그래밍 환경에서는 우리가 이것을 조절할 수 있습니다. 예를 들어, 증감 연산자를 사용할 때 아래 코드와 같이 두 가지 방법으로 사용할 수 있습니다.

```
//1번
var x = 0;
var z = ++x;
console.log(z); //1

//2번
var x = 0;
var z = x++;
console.log(z); //0
```

위 코드는 모두 변수 x의 값을 1 증가시키고 변수 z에 할당하는 코드입니다. 단, 1번 코드는 피연산자 변수보다 **연산자가 먼저** 나오는 전위 연산을 했기 때문에, x의 값을 증가시키는 작업을 변수 z에 값을 할당하기 전에 실행합니다. 하지만, 2번 코드의 경우엔 **연산자가 후에** 나오는 후위 연산을 했기 때문에, x의 값을 증가시키는 작업을 변수 z에 값을 할당한 후 실행합니다.

생각보다 이러한 사소한 코드의 차이가 결과의 많은 부분을 변화시킬 수 있다는 점을 항상 생각하고 코드를 작성하는 습관을 가지면 많은 도움이 될 수 있을 거라 생각합니다 :)

다음은, 할당과 연산의 관련된 내용입니다. 우리가 위에서 살펴본 코드는 피연산자 항이 기본적으로 두 개 이상이 나오는 이항 연산자입니다. 만약, 이미 선언한 변수에 산술 결과를 재할당하고 싶다고 가정할 경우 사용할 수 있는 방법이 **단항 연산자**를 사용하는 방법입니다.

연산자 이항 연산 표현 단항 연산 표현

+=	x = x + y	x += y
-=	x = x - y	x -= y
*=	x = x * y	x *= y
/=	x = x / y	x /= y
%=	x = x % y	x %= y

단항 연산자 사용법은 위 연산자 사용 방법에 대해서 이해하셨다면, 어렵지 않게 이해하실 수 있을 거라고 생각합니다. :)

산술 연산자의 마지막 내용으로 자바스크립트 코드를 작성할 때 굉장히 중요한 **문자열 결합**에 대해서 알아보도록 하겠습니다.

문자열 결합이란 이름 그대로 여러 문자열을 결합해서 하나의 문자열로 만들어내는 것을 말합니다. 문자열 결합은 굉장히 다양한 부분에서 사용될 수 있습니다. 문자열 결합은 숫자 연산과 마찬가지로 +, += 기호로 할 수 있습니다.

```
var str1 = "What a very "
str1 + "beautiful day"
console.log(str1); //What a very beautiful day

var str2 = "What a very"
"beautiful day " + str2
console.log(str2); //beautiful day What a very

var str3 = "What a very ";
str3 += "beautiful day";
console.log(str3); //What a very beautiful day
```

위 코드의 주요 포인트는 이미 선언된 문자열의 + 혹은 += 기호로 새로운 문자열을 만들어내는 데 있습니다. 단, "+" 기호로 문자열 결합을 실행할 경우 피연산자의 순서에 따라 문자열이 나타는 모양이 다르다는 점이 주요 포인트라고 할 수 있습니다.

또한, 문자열의 공백은 띄어쓰기를 나타내게 됩니다. 그렇기 때문에 문장을 만들 때 필요한 띄어쓰기를 위해서 문자열 끝에 공백을 넣거나 문자열 + " " + 문자열 형태로 의도적으로 공백을 추가할 수 있습니다.

비교 연산자

다음은 비교 연산자에 대한 내용입니다.

비교 연산자는 우리가 선언한 여러 변수를 비교하기 위해 사용되는 연산자입니다. 우리가 프로그램 코드를 작성하다 보면 변수의 값이 우리가 정의한 값과 같은 값인지 다른 값인지 비교하는 경우가 굉장히 많습니다.

이렇게 비교를 한다는 것은 비교 연산자는 참(true) 혹은 거짓(false)를 나타내는 Boolean 데이터 타입을 우리에게 돌려줍니다.

아래는 비교 연산자의 종류입니다.

연산자	의미	예
==	값 비교	x == y
===	엄격한 값 비교	x === y
!=	같지 않음	x != y
!==	엄격한 같지 않음	x !== y
<, >	크거나 작음	x > y, x < y
<=, >=	크거나 같음, 작거나 같음	x >= y, x<= y

마지막으로 강의에서 잠깐 다뤘던 삼항 연산자에 대해 알아보도록 하겠습니다. 삼항 연산자는 물음표(?) 기호와 콜론(:)으로 이루어지게 됩니다. 기본 형태는 아래와 같습니다.

```
변수 = 조건 ? 참 : 거짓;
```

```
var x = 0, y = 0
//조건에 따라 서로 다른 값을 할당
var z = x == y ? "참일 경우 할당할 값" : "거짓일 경우 할당할 값";
console.log(z); //참일 경우 할당할 값
```

삼항 연산자는 어떤 조건에 따라 서로 다른 값을 할당하고 싶을 때 자주 사용되는 연산자입니다. 지금 당장 코드가 익숙치 않더라도, 이렇게 코드를 작성 하는 방법이 존재한다는 것 정도만 기억해두셔도 필요할 때 찾아서 유용하게 사용할 수 있으실 겁니다. :)

사실, 연산자의 종류는 이 뿐만 아니라, 조건문에서 알아볼 논리 연산자, 비트 단위 연산이 가능한 비트 연산자가 더 존재합니다. 비트 단위 연산자의 경우에는 제대로 사용하면 굉장히 빠른 연산 속도를 자랑하는 좋은 연산자이지만, 난이도 있는 부분이 많습니다. 때문에 우리 강의에서는 다루지 않습니다. 그래도 혹시 내용이 궁금한 분들을 위해 [참고 사이트 (https://www.w3schools.com/js/js_bitwise.asp)](https://www.w3schools.com/js/js_bitwise.asp)를 첨부합니다.

내용을 적다보니 글이 길어졌습니다. 우리가 연산자를 사용한다는 것은 결국 우리가 원하는 어떤 논리적인 결과를 컴퓨터의 빠른 연산 속도를 이용해 대신 처리하길 바라기 때문에 사용한다고 할 수 있습니다. 컴퓨터는 굉장히 많은 양의 정보를 정말 빠르게 처리할 수 있습니다.

논리적인 구조를 생각하고 직접 코드로 작성해보는 연습을 많이 진행해보세요 :)