javascript

frontend 과정

연산자

- 용어정리
- 사칙연산, 거듭제곱과 나머지
- 대입 연산과 연산자의 반환값
- 복합할당 연산자
- 증감 연산자
- 연산자 우선순위
- 비교 연산자
- 논리 연산자
- typeof

용어정리

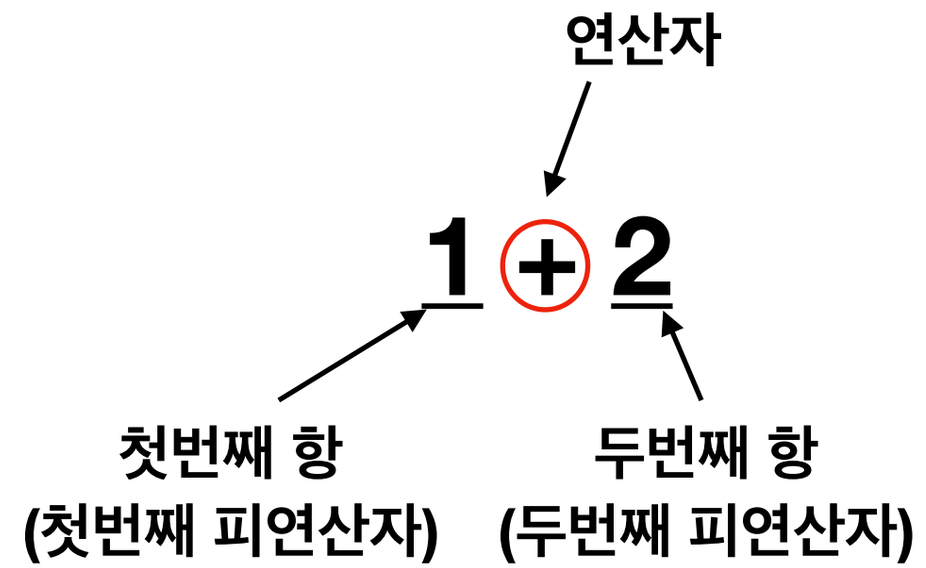
- 연산
  = 주어진 식을 계산하여 결과를 얻어내는 과정

- 연산자(operator)
  = 프로그래밍 언어로 하여금, 연산을 수행시키는 기호
  (항의 갯수나 자료형에 따라 동작이 달라질 수 있음에 주의)

- 피연산자 = 항 = 인수
  연산자가 연산을 수행하는 대상

- 단항, 이항, 삼항, ...
  연산자가 필요로하는 항의 갯수

- 표현식(expression)
  값을 반환(return) 하는 자바스크립트 코드



이항 연산자 +를 사용하여 3을 반환하는 표현식

사칙연산

- 두개의 숫자항을 대상으로 +, -, *(곱셈), / (나눗셈) 사용가능

```
const one = 1;
const two = 2;
const three = one + two;
console.log('three is', three);
const six = two * three;
console.log('six is', six);
```

```
three is 3
```

```
six is 6
```

```
console.log(three / two);
```

```
1.5
```

= 단항 연산자

- 두개의 항을 대상으로 = (대입연산자) 를 사용하는 경우
  왼쪽 피연산자에 오른쪽 피연산자의 값을 대입

```
const one = 1;
const two = 2;
const three = one + two;
```

- one + two 라는 식이 3을 반환하기 때문에
  three에 3을 대입

- 모든 연산자는 반환(return)값을 가짐
  (one + two가 3을 반환했듯이)

- 대입연산자의 반환값은 무엇일까요?

+ 단항 연산자

- + 연산자를 단항으로 사용하는 경우
  피연산자를 숫자 자료형으로 바꾸어 반환

```
a = new Date();
Thu Jul 07 2022 06:25:22 GMT+0900 (한국 표준시)
+a;
1657142722035
a = 'asdf';
'asdf'
+a;
NaN
a = '5';
'5'
+a
5
```

- 단항 연산자

- - 연산자를 단항으로 사용하는 경우
  +연산자 처럼 숫자 자료형으로 형변환시키고 음수로 반환

```
a = 'asdf';
'asdf'
```

```
-a
NaN
```

```
a = '5';
'5'
```

```
-a;
-5
```

거듭제곱, 나머지

- 두개의 숫자항을 대상으로 거듭제곱(**), 나머지(%) 사용가능

```
const a = 10;
undefined
```

```
a ** 1
10
```

```
a ** 2
100
```

```
a ** 3
1000
```

```
const a = 10;
undefined
```

```
a % 3;
1
```

연습문제

아래 표현식들의 결과를 예측해 보세요.

```
"" + 1 + 0
"" - 1 + 0
true + false
6 / "3"
"2" * "3"
4 + 5 + "px"
"$" + 4 + 5
"4" - 2
"4px" - 2
7 / 0
"  -9  " + 5
"  -9  " - 5
null + 1
undefined + 1
" \t \n" - 2
```

복합 대입 연산자

- 앞서 배운 -, +, *, /, %, ** 를 = 와 함께 사용하는 경우 복합 대입연산자를 사용하여 간결하게 표현 가능

```
a = 1;
1
a = a + 5;
6
```

```
a = 1;
1
a += 5
6
```

- Question 1. 좌측 이미지에선 연산자를 총 몇개 사용했을까요?
- Question 2. 좌측 이미지에선 항이 총 몇개일까요?
- Question 3. 우측 이미지에선 연산자를 총 몇개 사용했을까요?
- Question 4. 우측 이미지에선 항이 총 몇개일까요?

연습문제

아래 코드가 실행되고 난 후, a와 x엔 각각 어떤 값이 저장될까요?

```
let a = 2;

let x = 1 + (a *= 2);
```

증감 연산자

- 증가 (++), 감소(--) 단항연산자를 사용하여
  1을 간결하게 더하거나 뺄 수 있음 (숫자 형변환 일어남)

```
a = 1;
1
```

```
++a;
2
```

```
a
2
```

```
--a;
1
```

```
a
1
```

```
const b = --a;
```

```
c = 'Hello, World';
'Hello, World'
```

```
++c;
NaN
```

```
c = true;
true
```

```
++c;
2
```

```
c = false;
false
```

```
++c;
1
```

연습문제

아래 코드가 실행된 후, 변수 a, b, c, d엔 각각 어떤 값들이 저장될까요?

```
let a = 1, b = 1;

let c = ++a; // ?
let d = b++; // ?
```

연산자 우선순위

- 모든 연산자는 우선순위를 가짐
- 다음 이미지에서 연산자는 총 몇개 사용되었을까요?
- 각 연산자의 우선순위를 높은(먼저 연산되는) 순으로 나열해보세요

```
one = 1 + 2 * 3;
7
```

```
one
7
```

비교연산자

- 두개의 항을 받아 비교하고 boolean을 반환
- less than, greater than: <, >
- less than equal, greater than equal: <=, >=
- equal, not equal: ==, !=
- strict equality: ===, !==

```
a = 1; b = 2; c = '1', d = true;
```

```
a > 1;
false
```

```
a >= 1;
true
```

```
a < b
true
```

```
a == b
false
```

```
hello = 'hello';
'hello'
```

```
world = 'world';
'world'
```

```
hello < world;
true
```

```
hello > world;
false
```

```
a != b
true
```

```
a == c
true
```

```
a == d
true
```

```
a === c
false
```

```
a !== c
true
```

논리 연산자

- or: ||
- and: &&
- not: !

```
true || false
true
```

```
true && false;
false
```

&&는 false 표현식을 만나면
논리 연산을 종료하고 해당 표현식의 값을 반환

```
undefined && false;
undefined
```

```
undefined || false;
false
```

```
false || 'a' || true
'a'
```

|| 는 첫번째 true 표현식을 만나면
논리 연산을 종료하고 해당 표현식의 값을 반환

```
false && undefined;
false
```

```
!true
false
```

```
!false
true
```

typeof

- 모든 연산자가 기호로 되어있는건 아님
- typeof 라는 단항연산자는 우측 피연산자의 자료형을 반환
- (주의) null은 object가 아니지만 typeof null은 object를 반환

```
const a = 1;
undefined
typeof 'a';
'string'
typeof a
'number'
```

```
typeof null
'object'
typeof undefined
'undefined'
typeof NaN
'number'
```

연습문제

아래 표현식들의 결과를 예측해보세요.

```
1  5 > 4
2  "apple" > "pineapple"
3  "2" > "12"
4  undefined == null
5  undefined === null
6  null == "\n0\n"
7  null === +"\n0\n"
```

읽어볼 거리

- **연산자 우선순위표**
  (외워두면 편함)

- 비트와이즈 연산자, 쉼표 연산자
  (활용도가 비교적 낮아 강의에선 제외하였으나 알아두면 종종 쓰임)

- javascript에서 typeof null이 object를 반환하는 이유