# 5장 표현식과 문

4장에서는 '값'이라는 용어가 자주 등장했다. '값'이라는 용어를 알고 있다고 생각한다면 오산이다.

막상 설명하려면 난감할 수 있다. **개념을 이해한다는 것은 바로 용어를 정확히 이해하고 설명할 수 있다는 것이다.**

## 값

**값(value)은 표현식 (expression)이 평가(evaluate)되어 생성된 결과**를 말한다.

평가란 식을 해석해서 값을 생성하거나 참조하는 것을 의미한다.

`값 : 표현식이 평가되어 생성된 결과`

`10 + 20 = 30;` → 10+20은 평가되어 숫자 값 30을 생성한다.

4장에서 변수는 **하나의 값**을 저장하기 위해 확보한 메모리 공간 자체 또는 그 메모리 공간을 식별하기 위해 붙인 이름이라고 했다.

따라서 변수에 할당되는 것은 값이다.

```
var sum = 10 + 20; // 10+20이 평가되어 30이 할당된다.
```

위 코드에서 sum 에 할당되는 것은 10 + 20이 아니라, 10 + 20이 평가된 결과인 숫자 값 30이다.

값은 다양한 방법으로 생성할 수 있다. 위 코드처럼 **식으로 생성할 수도 있지만 가장 기본적인 방법은 리터럴을 사용하는 것이다.**

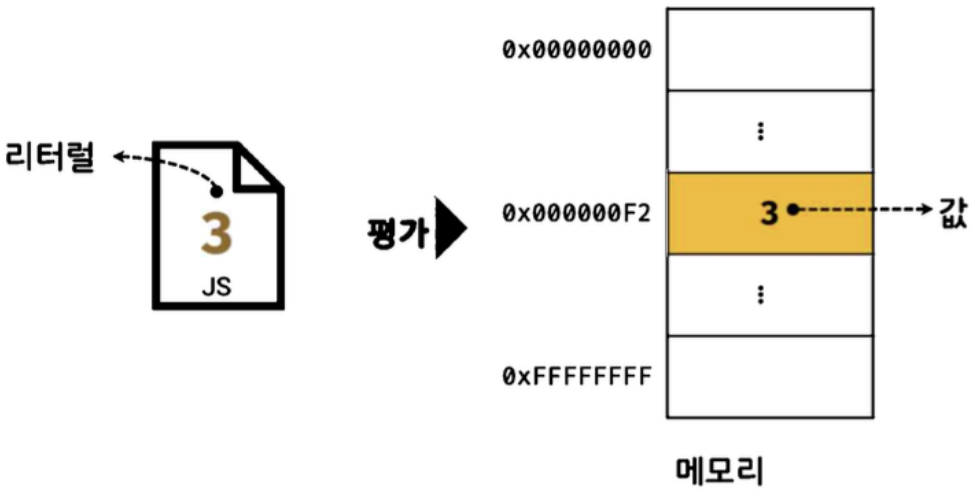
# 리터럴

**리터럴(literal)은 사람이 이해할 수 있는 문자 또는 약속된 기호를 사용해 값을 생성하는 표기법(notaion)을 말한다.**

```
3 -> 숫자 리터럴 3, 말 그대로 숫자 3을 의미함
```

위 코드의 3은 단순한 아라비아 숫자가 아니라 숫자 리터럴이다.

사람이 이해할 수 있는 아라비아 숫자를 사용해 숫자 리터럴 3을 코드에 기술하면 자바스크립트 엔진은 이를 평가해서 숫자 값 3을 **생성한다.**




자바스크립트 엔진은 코드가 실행되는 시점인 **런타임에** 리터럴을 평가해 값을 생성한다.

# 표현식

**표현식(expression)은 값으로 평가될 수 있는 문(statement)이다. 즉, 표현식이 평가되면 새로운 값을 생성하거나 기존 값을 참조한다.**

위에서 본 리터럴도 값으로 평가된다. 따라서 리터럴도 표현식이다.

`var score = 50+50;`

50+50은 리터럴과 연산자로 이뤄져 있다. 하지만 50+50도 평가되어 숫자 값 100을 생성하므로 표현식이다.

`score; // 100`

변수 score을 참조해보자. 변수 식별자를 참조하면 변수 값으로 평가된다.

**식별자 참조는 값을 생성하지는 않지만, 값으로 평가되므로 표현식이다.**

표현식은 리터럴,식별자(변수, 함수 등의 이름), 연산자, 함수 호출 등의 조합으로 이뤄질 수 있다.

```
10
'hello'
sum
person.name
arr[1]
10+20
sum = 10
sum !== 10
```

위와 같이 다양한 표현식이 있지만 값으로 평가된다는 점에서 모두 동일하다.

**즉, 값으로 평가될 수 있는 문은 모두 표현식이다.**

- `값으로 평가 됨? ㅇㅇ → 표현식`

표현식은 값으로 평가된다. 이때 표현식과 표현식이 평가된 값은 동등한 관계, 즉 동치(equivalent)다.

자바스크립트의 표현식 1+2는 평가되어 값 3을 생성하므로 표현식 1+2와 값 3은 동치다.

- 따라서 표현식은 값처럼 사용할 수 있으며, 이는 문법적으로 값이 위치할 수 있는 자리에는 표현식도 위치할 수 있다는 것을 의미한다.

산술 연산자 +의 좌항과 우항에는 숫자 값이 위치해야 한다.

이때 숫자 값으로 평가될 수 있는 표현식이라면 숫자 값 대신 사용할 수 있다.

```
var x = 1 + 2;
x+3; // 6
```

# 문

문과 표현식을 구별하고 해석할 수 있다면 자바스크립트 엔진의 입장에서 코드를 읽을 수 있고 실행 결과를 예측하는 데 도움이 된다.

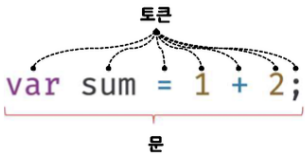
**문(statement)은 프로그램을 구성하는 기본 단위이자 최소 실행 단위이다.**

문의 집합으로 이뤄진 것이 바로 프로그램이며, 문을 작성하고 순서에 맞게 나열하는 것이 프로그래밍이다.

문은 여러 토큰으로 구성된다.

**토큰이란, 문법적인 의미를 가지며, 문법적으로 더 이상 나눌 수 없는 코드의 기본 요소를 의미한다.**

예를 들어, 키워드, 식별자, 연산자, 리터럴, 세미콜론 이나 마침표 등의 특수기호는 문법적인 의미를 가지며, 문법적으로 더 이상 나눌 수 없는 기본 요소이므로 토큰이다.




문은 명령문이라고도 부른다. 다른 말로하면 문은 컴퓨터에게 내리는 명령이다.

선언문, 할당문, 조건문, 반복문 등으로 구분할 수 있다.

```
var x; // 선언문

x = 5; // 할당문

function foo() {} // 함수 선언문

if(x>1) {console.log(x);} // 조건문

for (var i=0; i<2; i++) { console.log(i);} // 반복문
```

# 세미콜론과 세미콜론 자동 삽입 기능

세미 콜론은 문의 종료를 나타낸다.

자바스크립트 엔진은 세미콜론으로 문이 종료한 위치를 파악하고 순차적으로 하나씩 문을 실행한다.

**문의 끝에 붙이는 세미콜론은 옵션**이다. (세미콜론은 생략 가능하다.)

이는 자바스크립트 엔진이 소스 코드를 해석할 때 **문의 끝이라고 예측되는 지점에 세미콜론을 자동으로 붙여주는 세미콜론 자동 삽입 기능** (ASI)이 암묵적으로 수행되기 때문이다.

# 표현식인 문과 표현식이 아닌 문

표현식은 문의 일부일 수도 있고, 그 자체로 문이 될 수도 있다.

```
// 변수 선언문은 값으로 평가될 수 없으므로 표현식이 아니다.
var x;

// 1,2, 1+2, x=1+2는 모두 표현식이다.
x = 1 + 2;
// x = 1 + 2;는 표현식이면서 완전한 문이기도 하다.
```

이처럼 표현식과 문은 비슷해서 구별하기 어렵다고 느낄 수 있다.

하지만 표현식과 문을 구별하는 방법은 의외로 간단하다.

문에는 표현식인 문과 표현식이 아닌 문이 있다. **표현식인 문은 값으로 평가될 수 있는 문이며, 표현식이 아닌 문은 값으로 평가될 수 없는 문을 말한다.**

**표현식인 문과 표현식이 아닌 문을 구별하는 가장 간단하고 명료한 방법은 변수에 할당해보는 것이다.**

표현식인 문은 값으로 평가되므로 변수에 할당할 수 있다.

```
var foo = var x; // 표현식이 아닌 문은 값처럼 사용할 수 없다.

x = 100; // 할당문은 그 자체가 표현식이지만 완전한 문이기도 하다.
```

하지만 표현식이 아닌 문은 값으로 평가할 수 없으므로 변수에 할당하면 에러가 발생한다.

```
var foo = x = 100;

console.log(foo); // 100
```

할당문을 값처럼 변수에 할당했다.

표현식인 문인 할당문은 할당한 값으로 평가된다. 즉 `x=100`은 변수에 할당한 값 100으로 평가된다.

따라서 foo 변수에는 100이 할당된다.

# 요약

- **값 : 표현식이 평가되어 생성된 결과**
- **표현식 : 값으로 평가될 수 있는 문**
- **문 : 프로그램을 구성하는 기본 단위이자 최소 단위**
    - 표현식인 문
    - 표현식이 아닌 문 으로 나누어진다.