# Javascript 기초

## 반복문과 함수

김태민

# 저번에 우리는

# 연산자



## 산술 연산자

덧셈, 뺄셈, 곱셈, 나눗셈과 같은 수학 연산을 수행하는 연산자

| 구분 | 연산자 | 예 | 설명 |
|---|---|---|---|
| 이항 산술 | + | x + y | x에 y를 더합니다. |
| | - | x - y | x에서 y를 뺍니다. |
| | * | x * y | x에 y를 곱합니다. |
| | / | x / y | x를 y로 나눕니다. |
| | % | x % y | x를 y로 나누어 나머지를 구합니다. |
| | ** | x ** y | x의 y 거듭제곱을 구합니다. |
| 단항 산술 | ++ | x++ (후치 연산)<br>++x (전치 연산) | x를 1 증가시킵니다. |
| | -- | x-- (후치 연산)<br>--x (전치 연산) | x를 1 감소시킵니다. |
| 단항 부정 | - | -x | x의 부호를 부정합니다(음수 → 양수, 양수 → 음수) |

# 연산자



## 대입 연산자

데이터를 대입(할당)하는 연산

## 복합대입 연산자

산술 연산자와 대입 연산자를 함께 사용해 산술과 할당을 한 번에 수행

| 연산자 | 예 | 설명 |
|---|---|---|
| = | x = y | x에 y를 대입합니다. |
| += | x += y | x에 x + y를 대입합니다. |
| -= | x -= y | x에 x - y를 대입합니다. |
| *= | x *= y | x에 x * y를 대입합니다. |
| /= | x /= y | x에 x / y를 대입합니다. |
| %= | x %= y | x에 x % y를 대입합니다. |
| **= | x **= y | x에 x ** y를 대입합니다. |

# 연산자



## 비교 연산자

피연산자를 비교한 뒤,
논리형 값인 참(true),
거짓(false)을 반환

```
10 == '10'; // true
10 === '10'; // false
10 != '10'; // false
10 !== '10'; // true
10 < 10; // false
10 <= 10; // true
10 > 10; // false
10 >= 10; // true
```

| 연산자 | 예 | 설명 |
|---|---|---|
| == | x == y | x와 y의 값이 같으면 true를 반환합니다. |
| === | x === y | x와 y의 값과 자료형이 같으면 true를 반환합니다. |
| != | x != y | x와 y의 값이 다르면 true를 반환합니다. |
| !== | x !== y | x와 y의 값과 자료형이 다르면 true를 반환합니다. |
| 〈 | x 〈 y | x가 y보다 작으면 true를 반환합니다. |
| 〈= | x 〈= y | x가 y보다 작거나 같으면 true를 반환합니다. |
| 〉 | x 〉 y | x가 y보다 크면 true를 반환합니다. |
| 〉= | x 〉= y | x가 y보다 크거나 같으면 true를 반환합니다. |

## 논리 연산자

피연산자를 논리적으로 평가한 뒤, 조건에 맞는 피연산자를 반환

| 연산자 | 예 | 설명 |
| --- | --- | --- |
| && | x && y | x가 참이면 y를 반환하고, 거짓이면 x를 반환합니다. |
| \|\| | x \|\| y | x가 참이면 x를 반환하고, 거짓이면 y를 반환합니다. |
| ! | !x | x가 참이면 false를 반환하고, 거짓이면 true를 반환합니다. |

## 삼항 연산자

세 항 중 가장 왼쪽에 있는 피연산자의 참, 거짓에 따라
나머지 두 항에 있는 피연산자를 선택적으로 반환하는 연산

| 연산자 | 예 | 설명 |
| --- | --- | --- |
| ?: | x ? y : z | x가 참이면 y를 반환하고, x가 거짓이면 z를 반환합니다. |

```
let score = 90;
let grade = score >= 90 ? 'A+' : 'B';
console.log(grade); // A+
```

## If문 vs switch문

if 문은 조건에 식(statement)을 사용하고,

switch 문은 조건에 값(value)을 사용

90부터 99까지 'A++ 학점'이라고 출력하는 코드

```
let score = 90;
if(score >= 90 && score < 100){
        console.log("A++ 학점");
}
```

```
let score = 90;
switch(score){
        case 90:
        (중략)
        case 98:
        case 99:
                console.log("A++ 학점");
                break;
        default:
                break;
}
```

# 실습 환경

# 실습환경



OS : Window / Mac

브라우저 : Chrome

에디터 : VS Code https://code.visualstudio.com/

VS Code 익스텐션 : ESLint, Prettier, HTML CSS Support, HTML to CSS autocompletion, Auto Rename Tag, Auto Close Tag, htmltagwrap

Git : git bash, github 가입

# 반복문

## 반복문 loop

조건문과 더불어 JS의 핵심 문법

지정한 조건이 참(true)으로 평가되는 동안 지정한 블록문을 반복해서 실행하는 문법

while, do…while, for문

## 반복문 – while 문

While문은 특정 조건을 만족하는 동안 블록문을 실행

```
while(조건식) {
    // 조건식이 참이면 실행
}
```

## 반복문 – while 문

1부터 5까지 출력하는 코드 / 1부터 9999까지 출력하는 코드

```
console.log(1);
console.log(2);
console.log(3);
console.log(4);
console.log(5);
```

```
console.log(1);
console.log(2);
console.log(3);
console.log(4);
console.log(5);
console.log(6);
console.log(7);
console.log(8);
console.log(9);
... ??
```

## 반복문 – while 문

1부터 5까지 출력하는 코드 / 1부터 9999까지 출력하는 코드

```
let num = 1;
while(num <= 9999){
    console.log(num);
    num++;
}
```

## 무한반복문

모든 반복문은 사용할 때 한 가지 주의할 부분

반복문의 조건이 계속 참으로 평가되어 반복문이 끝나지 않고 무한히 실행되는 것

num을 증가시키는 코드가 없다면 num에 할당된 숫자 1은 영원히 1이기 때문에 반복문은 종료되지 않음

```
let num = 1;
while(num <= 9999){
    console.log(num);
    num++; // 코드가 한 번 반복될 때마다 num 변수를 1씩 증가시킵니다.
}
```

## 반복문 – do … while 문

특정 조건이 참으로 평가되는 동안 do 다음에 오는 블록문을 반복 실행

```
do{
    // 블록문
}while(조건식);
```

```
do{
    console.log("무조건");
    console.log("한 번은 실행");
}while(false);
```

## 반복문 – for 문

횟수를 지정해 지정한 횟수가 끝날 때까지 블록문을 반복 실행하는 반복문

초깃값 → 조건식 → 블록문(조건식이 참일 경우) → 증감식 → 조건식 순서로 실행

초깃값은 최초 1회만 실행

```
for(let i = 0; i < 5; i++){
    console.log(i);
}
```

## 반복문 – for 문

```
for(let i = 0; i < 5; i++){
    console.log(i);
}
```

| 순서 | 설명 |
|---|---|
| 1 | 반복문이 실행될 때 변수 i의 값을 0으로 초기화합니다. |
| 2 | 변수 i의 값이 5보다 작은지 평가합니다. |
| 2-1 | 참이면 3번으로 갑니다. |
| 2-2 | 거짓이면 반복문을 종료합니다. |
| 3 | 블록문을 실행합니다. |
| 4 | 블록문을 실행한 후 변수 i의 값을 1 증가시킵니다. |
| 5 | 변수 i가 5보다 작은지 다시 평가합니다. |
| 6 | 평가 결과가 거짓일 때까지 2번부터 4번을 반복합니다. |

## 반복문 – for 문

중첩 반복문의 기본이 되는 반복문은 가장 외부에서 실행되는 for 문
내부에 중첩된 for 문은 외부 for 문의 실행이 종료되면 같이 종료
가장 외부에 있는 for 문의 초깃값은 반복문이 실행될 때 1번만 설정
내부 반복문의 초깃값은 외부 블록문이 실행될 때마다 새로 설정

```
for(let i = 0; i < 2; i++){
    console.log(`i: ${i}`);
    for(let k = 0; k < 2; k++){
        console.log(`k: ${k}`);
    }
}
```

# 반복문

## 반복문 – for 문

```
for(let i = 0; i < 2; i++){
    console.log(`i: ${i}`);
    for(let k = 0; k < 2; k++){
        console.log(`k: ${k}`);
    }
}
```

| 순서 | 설명 |
|---|---|
| 1 | 외부 for 문의 초깃값을 설정합니다. |
| 2 | 외부 for 문의 조건식을 평가합니다. |
| 2-1 | 참이면 3번으로 갑니다. |
| 2-2 | 거짓이면 for 문을 종료합니다. |
| 3 | 외부 블록문을 실행합니다. |
| 4 | 내부 for 문의 초깃값을 설정합니다. |
| 5 | 내부 for 문의 조건식을 평가합니다. |
| 5-1 | 참이면 6번으로 갑니다. |
| 5-2 | 거짓이면 내부 for 문을 종료합니다. |
| 6 | 내부 블록문을 실행합니다. |
| 7 | 내부 for 문의 증감식을 실행합니다. |
| 8 | 내부 for 문의 조건식이 참일 동안 5번부터 7번까지 반복합니다. |
| 9 | 내부 for 문이 종료되면 외부 for 문의 증감식을 실행합니다. |
| 10 | 외부 for 문의 조건식이 참일 동안 2번부터 9번까지 반복합니다. |

## 반복문 – for 문과 배열

배열과 같은 자료형을 반복 횟수 용도로 자주 사용

for 문으로 배열 요소에 접근해 값을 출력

배열에 length 속성을 사용하여 배열의 데이터 개수(배열의 길이)를 확인

배열의 length 속성과 인덱스를 조합하여 배열 안의 요소가 몇 개든지 상관없이 반복하며 모든 요소에 접근

```
let arr = ["banana", "apple", "orange"];
for(let i = 0; i < arr.length; i++){
    console.log(arr[i]);
}
```

## 반복문 – for … in

ES6에서는 객체 리터럴이나 배열에 반복 접근할 수 있는 반복문이 몇 가지 추가

for 문의 소괄호 안에 in 키워드를 두고 키워드의 오른쪽에는 탐색의 대상이 되는 배열 또는 객체 리터럴

왼쪽에는 배열 또는 객체 리터럴을 탐색해서 키를 저장할 가변수(임시 변수)

```
for(가변수 in 배열/객체 리터럴){
    // 블록문
}
```

## 반복문 – for … in : 객체 리터럴 반복

탐색 결과로 가변수에 객체 리터럴의 키가 할당되어 객체 리터럴의 키와 값을 출력

```
let obj = {name:"철수", age:"20"};
for(let key in obj){
    console.log(key + ": " + obj[key]);
}
```

## 반복문 – for … in : 배열 반복

for...in 문으로 반복할 때 배열의 순서대로 접근하는 것을 보장하지 않으므로 코드를 작성할 때 주의

```
let arr = ["orange", "banana", "apple"];
for(let index in arr){
    console.log(index + ": " + arr[index]);
}
```

## Break 문

종료 조건을 만족하지 않아도 인위적으로 반복문을 종료하는 방법

어떤 반복문이라도 break 문을 만나면 반복문 종료

반복문 때문에 0부터 9까지 출력해야 하지만, 변수 i가 5가 되는 순간 break 문이 실행되어 반복문이 종료됨

```
for(let i = 0; i < 10; i++){
    console.log(i);
    if(i === 5) break;
}
```

## Break 문

for...in 문을 사용해도 마찬가지

가변수에 담기는 값이 age일 때 break 문이 실행되어 반복문 종료

```
let obj = {name:"철수", age:20};
for(let key in obj){
    if(key === "age") break;
    console.log(obj[key]);
}
```

## continue 문

break 문이 반복문을 즉시 종료하는 명령이라면, continue 문은 반복문을 건너뛰고 실행하라는 명령

반복문의 블록문에서 continue 문을 만나면 해당 반복 실행만 건너뜀

1부터 10까지 반복하는데, 홀수일 때는 반복문을 건너뛰고 짝수만 출력

```
for(let i = 1; i <= 10; i++){
  if(i % 2 === 1) continue;
  console.log(i);
}
```

## continue 문

break 문이 반복문을 즉시 종료하는 명령이라면, continue 문은 반복문을 건너뛰고 실행하라는 명령

반복문의 블록문에서 continue 문을 만나면 해당 반복 실행만 건너뜀

1부터 10까지 반복하는데, 홀수일 때는 반복문을 건너뛰고 짝수만 출력

```
for(let i = 1; i <= 10; i++){
  if(i % 2 === 1) continue;
  console.log(i);
}
```

# 자바스크립트 함수

## 함수

어떤 목적을 가지고 작성한 코드를 모아 둔 블록문

코드를 함수로 만들면 함수를 호출해 함수 내부에 모아 둔 여러 줄의 코드를 한 번에 실행 가능

구구단 3단을 계산하는 코드를 프로그램의 여러 군데에서 사용해야 한다.

```
for(let i = 1; i <= 9; i++){
    console.log(`3 * ${i} = ${3 * i}`);
}
```

## 함수

구구단 코드를 다음과 같이 묶으면 편리

블록문을 function 키워드, 식별자, 소괄호와 함께 묶으면 함수가 생성되는데, 이를 **함수를 정의**한다

필요할 때마다 코드를 새로 작성할 필요 없이 정의한 함수를 호출하기만 하면 됨

```
function gugudan(){ // 함수 시작
    for(let i = 1; i <= 9; i++){
        console.log(`3 * ${i} = ${3 * i}`);
    }
} // 함수 끝
```

## 함수정의 – 함수 선언문

function 키워드로 함수를 정의하는 방법

function 키워드 다음에 함수를 식별할 수 있도록 식별자를 붙이고 소괄호(())를 붙임

식별자의 명명 규칙은 변수명을 생성할 때와 같음

```
function gugudan(){
    for(let i = 1; i <= 9; i++){
        console.log(`3 * ${i} = ${3 * i}`);
    }
}
gugudan();  // 함수를 실행한다, 함수를 호출한다
```

## 함수정의 – 함수 표현식

함수는 객체에서 파생된 자료형. 자바스크립트에서 자료형은 변수에 할당할 수 있음

함수도 변수에 할당할 수 있음

앞에서 함수 선언문으로 정의한 gugudan() 함수를 함수 표현식으로 바꾸면 다음과 같음

```
const gugudan = function(){
    for(let i = 1; i <= 9; i++){
        console.log(`3 * ${i} = ${3 * i}`);
    }
};
gugudan(); // 함수 호출
```

## 함수정의 – 화살표 함수

ES6에서 추가된 함수 정의 방법

함수를 호출하려면 정의된 함수를 변수에 할당하는 방법인 함수 표현식을 사용해야 함

```
const gugudan = () => {
    for(let i = 1; i <= 9; i++){
        console.log(`3 * ${i} = ${3 * i}`);
    }
};
gugudan();
```

## 매개변수와 인수

3단이 아니라 5단, 8단처럼 다른 단을 출력하려면…

함수 내부의 블록문을 보면 빨간색 네모 안의 단수만 다르고 나머지는 동일함

```
function gugudan5(){
    for(let i=1; i<=9; i++){
        console.log(`5 * ${i} = ${5 * i}`);
    }
}
```

```
function gugudan8(){
    for(let i=1; i<=9; i++){
        console.log(`8 * ${i} = ${8 * i}`);
    }
}
```

## 매개변수와 인수

**매개변수**는 함수를 정의할 때 외부에서 전달하는 데이터를 함수에서 받을 수 있도록 정의하는 변수

정의한 함수를 호출할 때 소괄호 안에 전달하고 싶은 데이터를 적는데, 이를 **인수**라고 함

함수 호출 시 전달하는 데이터 즉, 인수는 함수의 매개변수에 자동으로 할당됨

```
// 함수 선언문
function 함수명(매개변수1, 매개변수2, ..., 매개변수N){}
// 함수 표현식
const 함수명 = function 식별자(매개변수1, 매개변수2, ..., 매개변수N){};
// 화살표 함수
const 함수명 = (매개변수1, 매개변수2, ..., 매개변수N) => {};
// 함수 호출
함수명(인수1, 인수2, ..., 인수N);
```

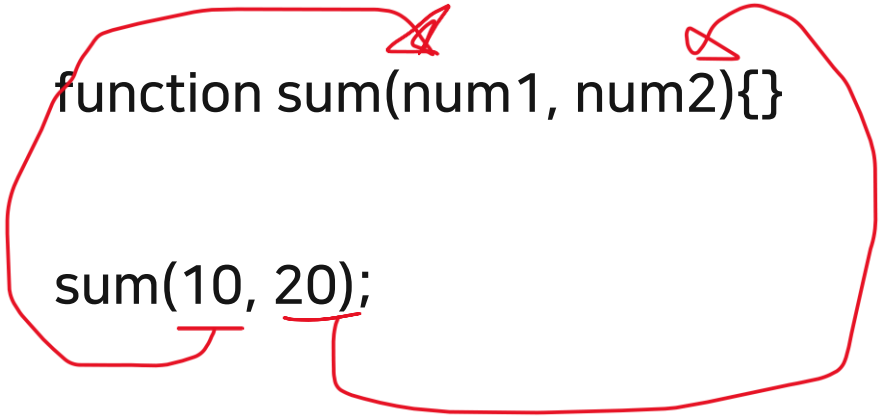
## 매개변수와 인수

```
function gugudan(① dan){
    for(let i = 1; i <= 9; i++){
        console.log(`${dan} * ${i} = ${dan * i}`);
    }
}
gugudan(② 3); // 3단 출력
gugudan(② 5); // 5단 출력
gugudan(② 8); // 8단 출력
```

① 매개변수 ② 인수

## 매개변수와 인수

함수를 정의할 때 함께 정의한 매개변수는 함수 호출 시 전달되는 데이터와
일대일로 대응되어 데이터가 할당됨
첫 번째 인수는 첫 번째 매개변수에 할당되고, 두 번째 인수는 두 번째 매개변수에 할당됨



## 매개변수와 인수

변수를 선언하고 값을 할당하지 않으면 undefined 값으로 초기화되는 것처럼
매개변수도 함수 호출 시 데이터를 전달하지 않으면 undefined 값이 할당되어
코드를 실행해도 오류가 발생하지 않음.

함수를 호출할 때 데이터를 전달하지만, 매개변수가 정의되지 않은 경우에도 오류는 안남

```
const sum = (num1, num2) => {
    console.log(num1, num2);
}
sum(10, 20);
```

```
const sum = num1 => {
    console.log(num1, num2);
}
sum(10);
```

## 매개변수와 인수 – 기본값 할당

ES6에서 기본값을 지정하는 방식이 추가됨

매개변수에 직접 데이터를 할당하는 방식으로 단순하게 기본값 지정

```
function sum(a = 10, b = 10){
    console.log(a, b);
}
sum(); // 10, 10
```

## 매개변수와 인수 – …rest

나머지 매개변수(Rest Parameters).

마치 "나머지 전부"를 싹 다 모아 담는 그릇 같은 역할

함수에 여러 개의 인자를 전달했는데, 그중 일부는 정해진 매개변수로 받고

나머지는 전부 하나의 배열로 묶어서 받고 싶을 때 …rest를 사용

```
function sum(first, second, ...others) {
    console.log(first);   // 첫 번째 인자
    console.log(second);  // 두 번째 인자
    console.log(others);  // 나머지 인자들이 배열로 묶여서 들어옴
}
sum(1, 2, 3, 4, 5);
```

## return문

함수 외부로 데이터를 반환할 때는 return 문을 사용

매개변수에 직접 데이터를 할당하는 방식으로 단순하게 기본값 지정

```
function sum(num1, num2){
  let result = num1 + num2;
  console.log("inner: " + result);
}
sum(10, 20); // inner: 30
```

```
function sum(num1, num2){
  let result = num1 + num2;
}
sum(10, 20);

// ReferenceError: result is not defined
console.log("out:" + result);
```

## return문

오류가 나지 않게 하려면 다음 코드와 같이 return 문으로 함수 내부 데이터를 함수 외부로 전달해야 함

sum 함수의 내부 변수인 result가 return 문에 작성

함수 내부 변수인 result에 할당된 값, 즉 데이터가 sum() 함수를 호출한 곳으로 전달됨.

이를 **반환한다**고 하며, 이때 반환된 데이터를 **반환값**이라고 함

```
function sum(num1, num2){
  let result = num1 + num2;
  return result;
}
const result = sum(10, 20);
console.log("out: " + result); // out: 30
```

## return문

데이터를 반환할 때 return 문에 꼭 변수를 사용해야 하는 건 아니고
다음처럼 표현식으로도 데이터를 반환할 수 있음

```
function sum(num1, num2){
  return num1 + num2;
}
const result = sum(10, 20);
console.log("out: " + result);
```

## return문

데이터를 반환하지 않으면 단순히 함수 실행을 종료하는 역할만 함.

함수 내부에서 데이터를 반환하지 않는 return 문을 만나면 return 문 다음에 코드가 있더라도 함수 실행을 즉시 종료하고 undefined를 반환함

```
function sum(num1, num2){
    if(typeof num1 !== "number" || typeof num2 !== "number"){
        return; // 매개변수가 숫자가 아니면 강제 종료
    }
    return num1 + num2;
}
let result = sum("a", "b");
console.log("out: " + result); // out: undefined
```

## return문

화살표 함수에서 {}를 생략하면 화살표 다음에 오는 코드는 return 문으로 처리

```
const sum = (num1, num2) => num1 + num2;
const result = sum(10, 20); // 30
```

## 생성자

객체를 만드는 방법

생성자 함수는 마치 붕어빵 틀처럼, 미리 정해진 형태의 객체를 찍어낼 수 있게 해주는 특별한 함수

함수 이름은 대문자로 시작! (관례적으로)

new 키워드와 함께 호출!

```
function Person(name, age) { ... }
const person1 = new Person('김태민', 32);
const person2 = new Person('김태구', 3);
const person3 = new Person('고승희', 30);
```

## 생성자

생성자 함수 내부에서 this는 새롭게 생성될 객체 자신을 지칭함

**new Person('김태민', 30)** 이렇게 호출하면, 자바스크립트 내부에서는 다음과 같은 일들이 일어납니다.

1. 빈 객체 생성: {} 이런 빈 객체가 하나 만들어집니다.
2. this 바인딩: 방금 만들어진 빈 객체가 생성자 함수 안의 this로 연결됩니다.
3. 속성 추가: **this.name = name;** 이런 식으로 this에 속성과 값을 추가하면, 새로 만들어진 객체에 담김.
4. 객체 반환: 생성자 함수가 실행을 마치면, this에 담겨 있던 객체가 최종적으로 반환됩니다.

생성자

```
function Person(name, age) {
    // 여기서 this는 새롭게 만들어질 객체!
    this.name = name; // 새로 만들어질 객체에 name 속성 추가
    this.age = age;   // 새로 만들어질 객체에 age 속성 추가
    this.greet = function() {
        console.log(`안녕하세요, 저는 ${this.name}입니다.`); // 여기서 this는 Person 객체 자신
    };
}
const person1 = new Person('김태민', 30);
console.log(person1.name); // 출력: 김태민
person1.greet(); // 출력: 안녕하세요, 저는 김태민입니다.
```

## 상황마다 다른 this

생성자 함수에서 this는 새로 만들어질 객체를 가리키는 것으로 고정되지만,
일반 함수에서 this는 함수가 어떻게 호출되었는지에 따라 달라짐

**일반 함수 호출: someFunction(); -> this는 window (브라우저) 또는 undefined (엄격 모드)**

**메서드 호출: obj.method(); -> this는 obj**

**이벤트 핸들러: element.addEventListener('click', function() { console.log(this); }); -> this는 element**

# 오늘 우리는

## 반복문 loop

조건문과 더불어 JS의 핵심 문법

지정한 조건이 참(true)으로 평가되는 동안 지정한 블록문을 반복해서 실행하는 문법

while, do…while, for문

## Break 문

종료 조건을 만족하지 않아도 인위적으로 반복문을 종료하는 방법

어떤 반복문이라도 break 문을 만나면 반복문 종료

반복문 때문에 0부터 9까지 출력해야 하지만, 변수 i가 5가 되는 순간 break 문이 실행되어 반복문이 종료됨

```
for(let i = 0; i < 10; i++){
    console.log(i);
    if(i === 5) break;
}
```

## continue 문

break 문이 반복문을 즉시 종료하는 명령이라면, continue 문은 반복문을 건너뛰고 실행하라는 명령

반복문의 블록문에서 continue 문을 만나면 해당 반복 실행만 건너뜀

1부터 10까지 반복하는데, 홀수일 때는 반복문을 건너뛰고 짝수만 출력

```
for(let i = 1; i <= 10; i++){
  if(i % 2 === 1) continue;
  console.log(i);
}
```

## 함수

어떤 목적을 가지고 작성한 코드를 모아 둔 블록문

코드를 함수로 만들면 함수를 호출해 함수 내부에 모아 둔 여러 줄의 코드를 한 번에 실행 가능

구구단 3단을 계산하는 코드를 프로그램의 여러 군데에서 사용해야 한다.

```
for(let i = 1; i <= 9; i++){
    console.log(`3 * ${i} = ${3 * i}`);
}
```

## 함수정의 – 화살표 함수

ES6에서 추가된 함수 정의 방법

함수를 호출하려면 정의된 함수를 변수에 할당하는 방법인 함수 표현식을 사용해야 함

```
const gugudan = () => {
    for(let i = 1; i <= 9; i++){
        console.log(`3 * ${i} = ${3 * i}`);
    }
};
gugudan();
```

## 매개변수와 인수

3단이 아니라 5단, 8단처럼 다른 단을 출력하려면…

함수 내부의 블록문을 보면 빨간색 네모 안의 단수만 다르고 나머지는 동일함

```
function gugudan5(){
    for(let i=1; i<=9; i++){
        console.log(`5 * ${i} = ${5 * i}`);
    }
}
```

```
function gugudan8(){
    for(let i=1; i<=9; i++){
        console.log(`8 * ${i} = ${8 * i}`);
    }
}
```

# 감사합니다