

## 04. 함수와 스코프

---

# 프로그래밍의 꽃, 함수

---

# 왜 함수를 사용할까

- 프로그래밍에서 가장 중요한 것은 문제를 분석하는 것
- 주어진 문제를 여러 개의 작은 문제로 나눈 후  
작은 문제를 하나씩 해결하면서 최종적으로 주어진 문제를 끝낸다.
- 가장 작은 단위로 나눈 것을 함수로 작성한다.

(예) 투두리스트

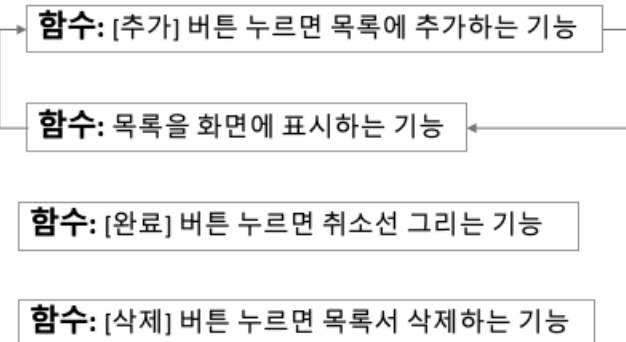
- 폼에 내용을 입력하고 [추가] 버튼을 클릭하면 추가한 내용이 화면에 표시된다.
- 내용 오른쪽에 있는 [완료] 버튼을 클릭하면 취소선이 그려지고
- [삭제] 버튼을 클릭하면 목록에서 삭제되도록 한다.

→ 함수 없이 작성한다면 입력 창에 내용이 입력될 때마다 같은 명령을 계속 반복해야 한다. 하지만 기능별로 함수를 따로 만들어 둔다면 필요한 함수별로 실행할 수 있다.

### 오늘의 할 일

함수 공부하기

연습문제 풀기



# 함수 선언 & 실행

- 함수를 선언할 때에는 function이라는 예약어를 사용하고
- 함수 이름을 적은 후 중괄호 안에 실행할 여러 명령들을 묶는다.
- 함수를 실행(호출)할 때는 함수 이름 뒤에 중괄호 ( )를 붙인다.

## 함수 선언

```
function 함수명() {  
    명령(들)  
}
```

## 함수 실행

```
함수명()
```

(예) 1부터 10까지 더하는 기능을 함수로 만들어서 실행하기

함수 선언

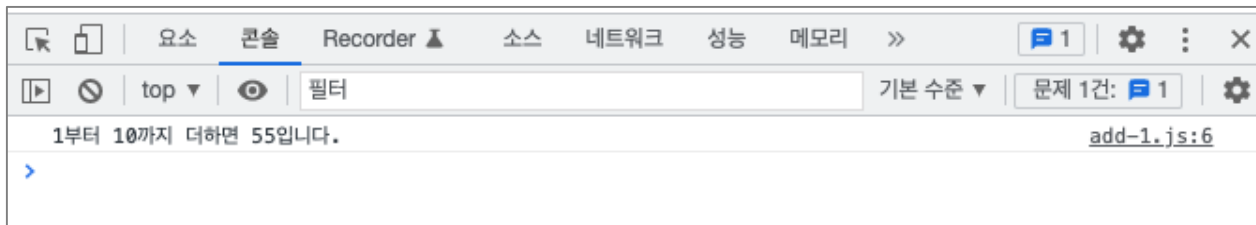
```
function calcSum() {  
  let sum = 0;  
  for(let i = 1; i <= 10; i++) {  
    sum += i;  
  }  
  console.log(`1부터 10까지 더하면 ${sum}입니다.`);  
}
```

함수 실행



```
calcSum();
```

함수 선언 소스는 어디에 넣어도 상관없다.  
하지만, 일반적으로 함수를 선언하는 소스를  
실행 소스보다 앞에 넣는다.



# 매개변수와 인수

앞에서 만들었던 calcSum 함수는 몇 번을 실행해도 1부터 10까지 더한 값만 보여 준다.

→ 같은 방법으로 1부터 50까지 더하려면? 1부터 100까지 더하려면

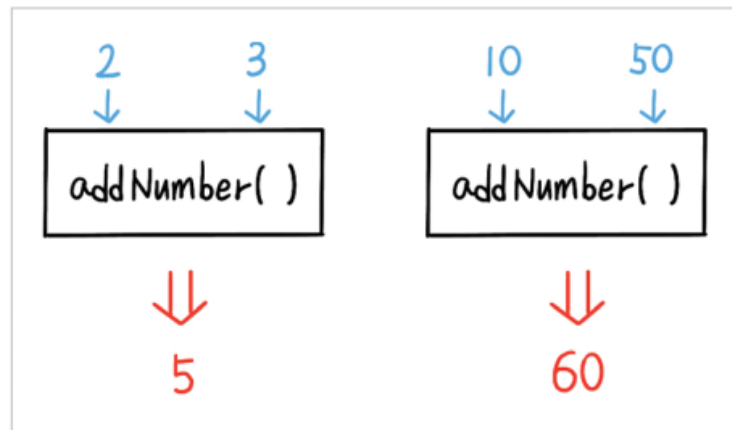
→ 앞에서 만든 calcNum 함수를 재사용할 수 있다.

→ 함수를 선언할 때, 함수를 실행할 때 변수를 사용하자!

```
<script>
  function addNumber(num1, num2)
{
  let sum = num1 + num2;
  alert("결과 값: " + sum);
}
  addNumber(1, 5);
</script>
```

함수를 여러 번 사용할 수 있게 해보자

← 항상 같은 값이 출력됨



# 매개변수와 인수

매개변수와 인수를 통틀어서 '인자'라고도 함

## 매개변수

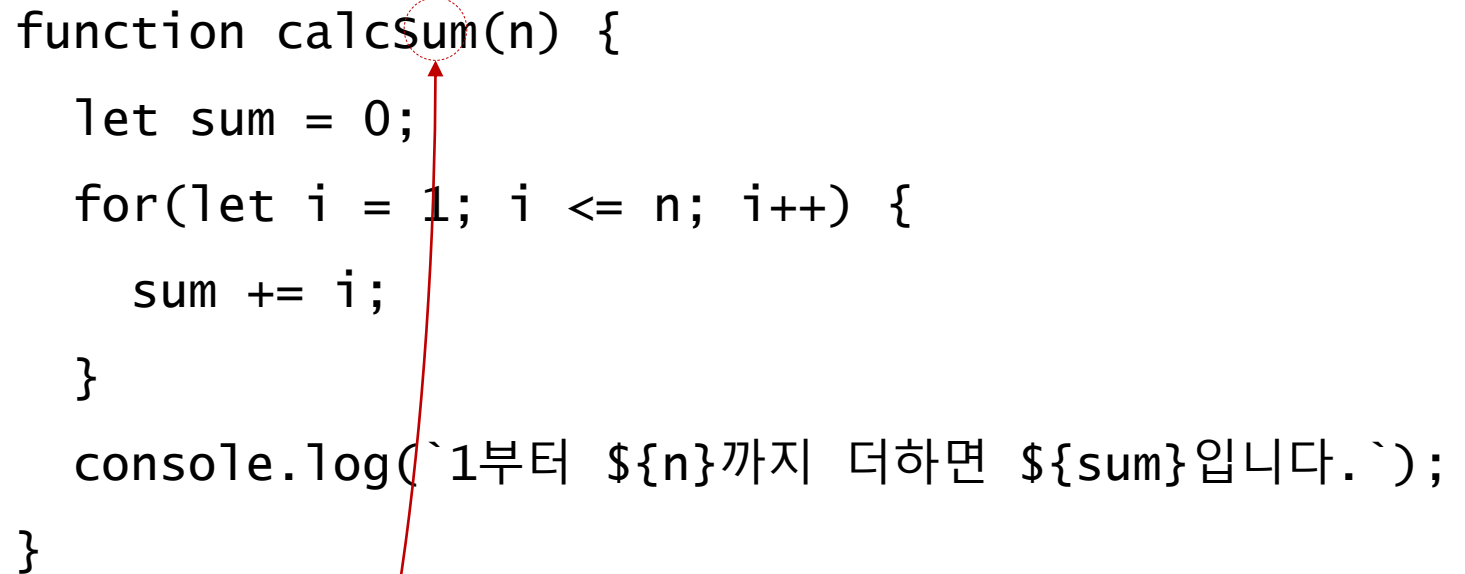
- 함수를 선언할 때 외부에서 값을 받는 변수
- 함수 이름 옆의 괄호 안에 매개변수 이름을 넣어준다
- 매개변수에 이름을 붙이는 방법은 일반적인 변수 이름을 붙이는 방법과 같다.
- 매개변수는 선언된 함수에서만 사용한다.
- 함수에 여러 개의 매개변수가 필요할 때에는 매개변수 사이에 쉼표(,)를 찍으면서 나열한다.

## 인수

매개변수가 있는 함수를 실행할 때, 매개변수로 값을 넘겨주는 변수

# 매개변수와 인수

1부터 n까지 더하는 함수 calcSum(n) 선언



```
function calcSum(n) {  
  let sum = 0;  
  for(let i = 1; i <= n; i++) {  
    sum += i;  
  }  
  console.log(`1부터 ${n}까지 더하면 ${sum}입니다.`);  
}
```

The code block is enclosed in a light gray box. A blue L-shaped line points from the text '1부터 n까지 더하는 함수 calcSum(n) 선언' to the 'function' keyword. A red curved arrow points from the '5' in 'calcSum(5);' to the 'n' parameter in the function signature.

1부터 5까지 더해라. calcSum(5)

calcSum(5);



# 자바스크립트에서 함수 처리 순서

```
(1) calcSum(10);  
  
function calcSum(n) {  
  let sum = 0;  
  for(let i = 1; i <= n; i++) {  
    sum += i;  
  }  
  console.log(`1부터 ${n}까지 더하면 ${sum}입니다.`);  
}
```

(2) 10

(3)

자바스크립트는 function이라는 예약어를 만나면 함수를 선언한다는 것을 알고 일단 메모리 어딘가에 calcSim() 함수의 내용을 저장해둔다.

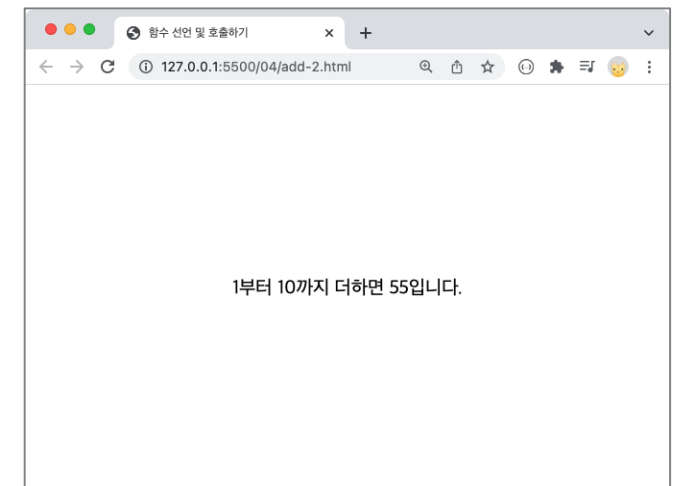
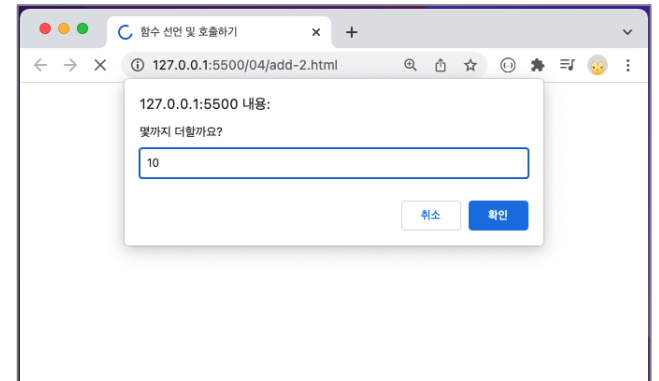
- 1) 함수 선언 다음에 오는 calcSum(10) 명령을 만나면 calcSum 함수를 실행한다.
- 2) 이 때 소괄호 안에 있는 num은 함수를 선언할 때 지정한 매개변수 n으로 넘겨진다.
- 3) 그 값을 사용해서 함수에 있던 명령을 차례대로 실행한다.

# return문

- 함수 안에서 실행하고 그 결과를 함수 밖에서 받아 처리해야 할 경우도 많다.
- 함수의 실행 결과를 함수를 실행한 시점으로 넘겨주는 것을 '결과값을 반환한다' `return` 라고 한다.
- 함수를 실행한 후 결과를 반환할 때는 예약어 `return` 다음에 넘겨줄 값이나 변수를 지정한다.

```
function calcSum(n) { // n: 매개변수
    let sum = 0;
    for(let i = 1; i <= n; i++) {
        sum += i;
    }
    return sum;
}

let num = parseInt(prompt("몇까지 더할까요?"));
document.write(`1부터 ${num}까지 더하면 ${calcSum(num)}입
니다.`);
```



# 기본 매개변수

ES6에는 기본 매개변수가 있어서 함수를 정의할 때 매개변수의 기본값을 지정할 수 있다.

→ 함수를 실행할 때 인수가 부족하면 기본값을 사용한다.

```
function multiple(a, b = 5, c = 10 ) {  
    return a * b + c;  
}  
  
multiple(5, 10, 20)  
multiple(10, 20)  
multiple(10);
```

# [실습] 개발자 도구창의 디버깅 기능

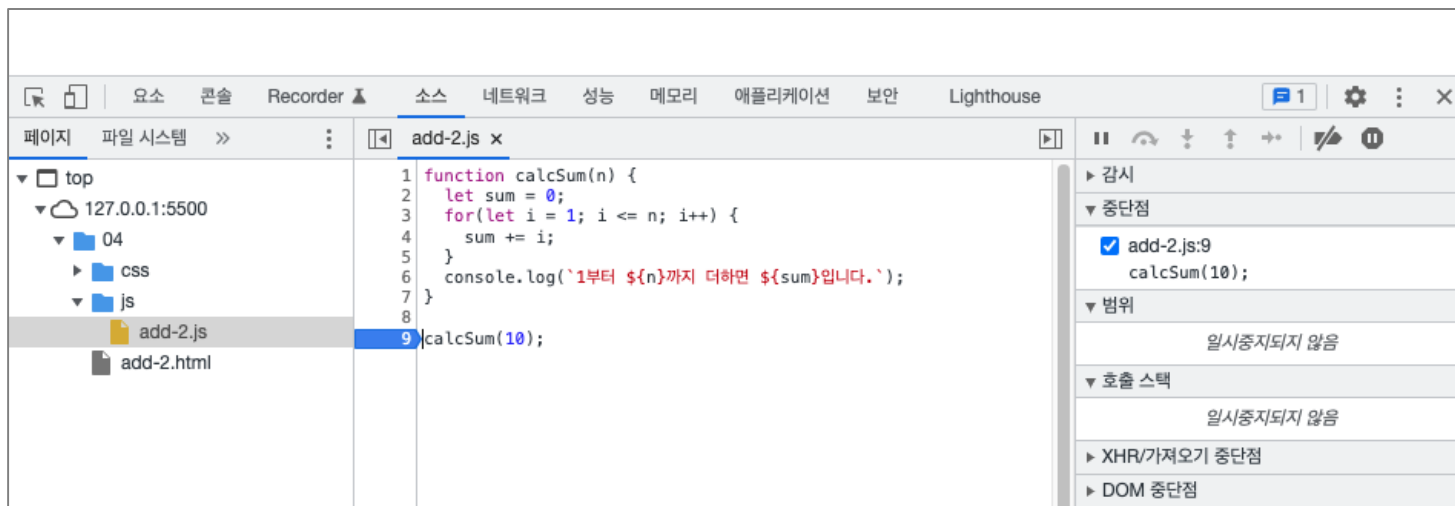
**디버깅(debugging)** : 프로그램의 결과가 예상했던 것과 다르게 나왔을 때 순서대로 하나씩 진행해 보면서 오류를 찾아내는 과정. (버그를 찾는 과정)

웹 개발자 도구 창에는 '디버깅' 기능이 포함되어 있어서 프로그램을 만들면서 오류가 발생했을 때 어디에서 문제가 발생했는지 찾아볼 수 있다.

개발자 도구 창의 디버깅 기능을 사용해서, 프로그램 안에서 소스가 어떻게 동작하는지, 변수에 값이 제대로 할당되는지 등을 눈으로 확인하고 오류를 찾아낼 수 있다.

# [실습] 개발자 도구창의 디버깅 기능

- 1) VS Code에서 04Wadd-2.html 파일을 열고 마우스 오른쪽 버튼을 클릭한 후 [Open with live server] 선택
- 2) 개발자 도구 창에서 [소스] 탭을 클릭한 후, js 폴더 앞에 있는 ►를 클릭하고, add-2.js 선택
- 3) 소스에서 중간 실행 결과값이나 변수값을 확인하려면 그 위치에 표시 → 브레이크포인트 또는 중단점이라고 한다.
- 4) 여기에서는 calcSum(10) 명령 왼쪽의 줄번호 '9' 클릭해서 중단점 만든다.

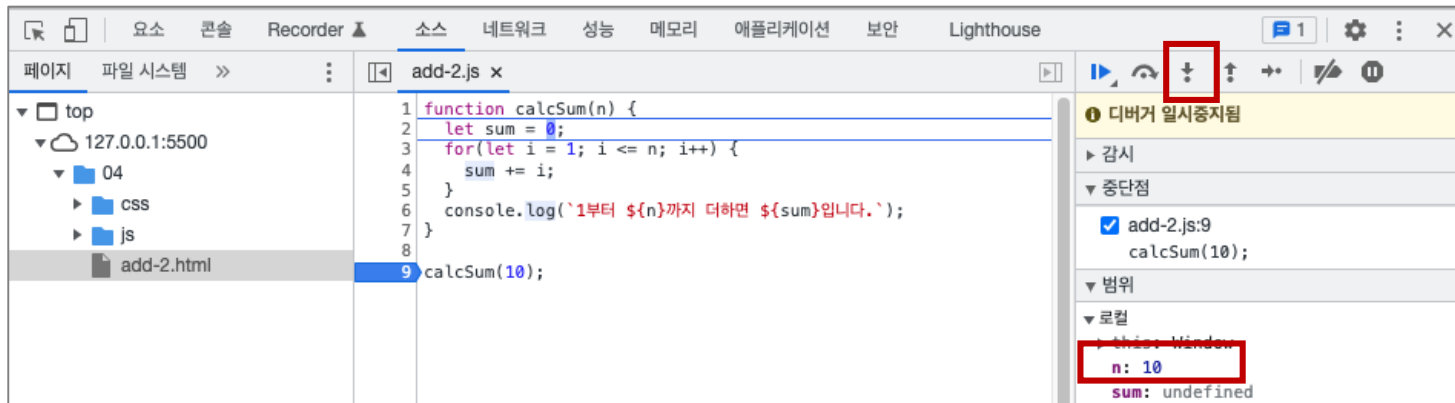
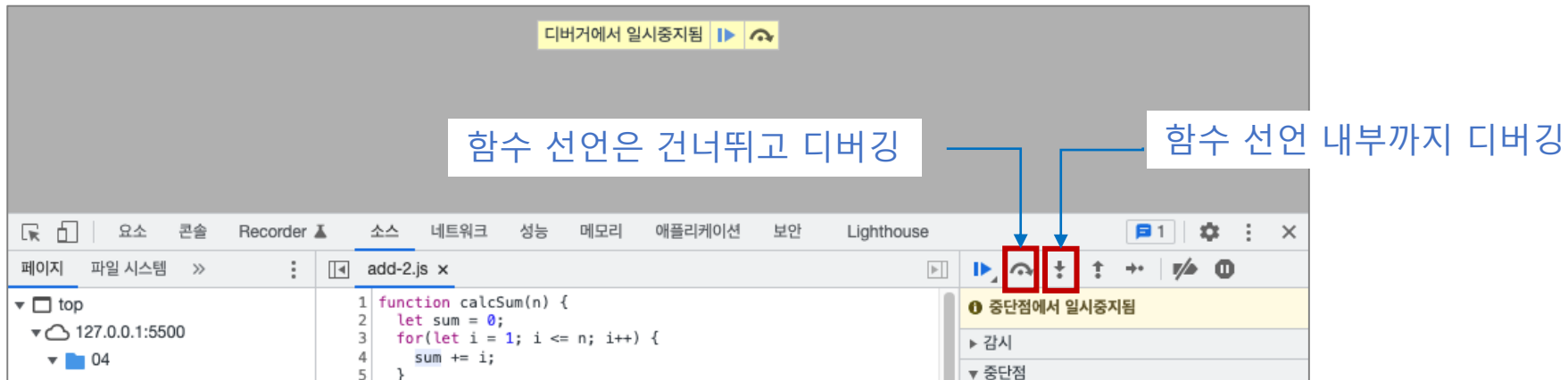



5) 브레이크포인트를 지정한 후에는 소스를 다시 실행해야 한다. 브라우저 새로고침 클릭

6) '디버거에서 일시중지됨'이라는 메시지와 함께 디버깅을 시작할 준비 끝.


7)  클릭

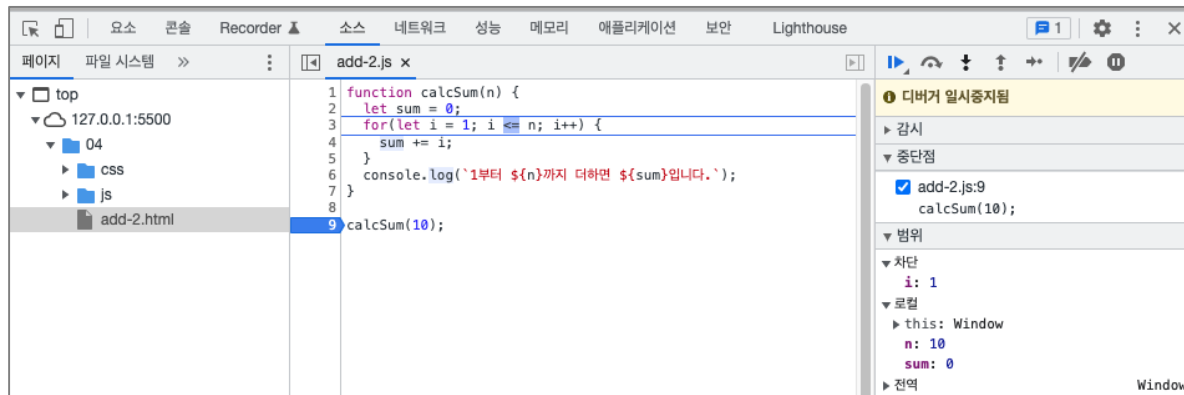
8) calcSum 함수로 넘어가면서 n에 10이라는 인수를 넘겨준다



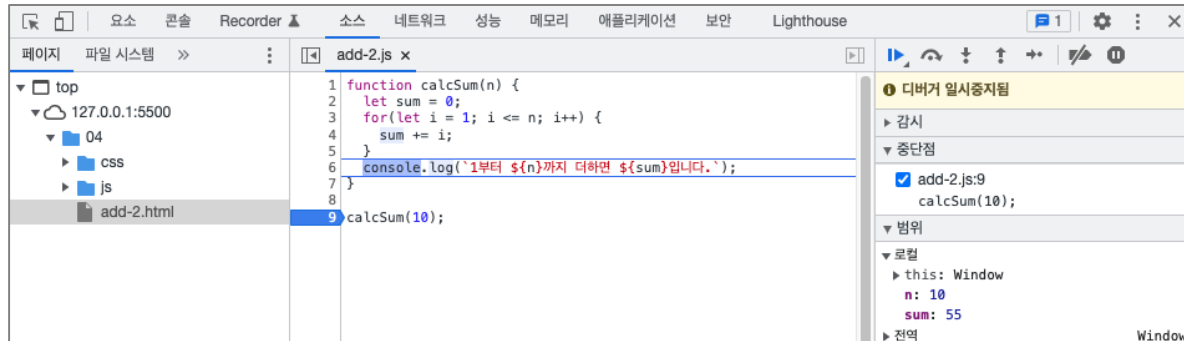
9) 다시 한번  클릭

10) sum 변수에 값이 할당되면서 드디어 for문으로 들어간다.

11)  를 클릭할 때마다 소스가 한 줄씩 처리되고, 화면 오른쪽에 변수들의 값이 바뀌어 나타난다.  
i 값을 1 증가시킨 후 조건을 비교하는 과정을 하나하나 눈으로 확인할 수 있습니다.



12) 마지막에 i 값이 11이 되면  $i \leq 10$  조건에 맞지 않으므로 for문을 빠져나오고 콘솔 창에 결과를 표시한다.



---

# 변수의 유효 범위, 스코프

---



# 스코프

스코프(scope) : 선언한 변수의 적용 범위를 가리킨다.

- var 키워드를 사용한 변수는 함수 레벨 스코프
- let, const 키워드를 사용한 변수는 블록 레벨 스코프

# var 변수의 스코프 - 함수 스코프

- var 예약어를 사용해서 변수를 선언하면 해당 변수는 함수 스코프를 가진다.
- 함수 스코프란 변수를 선언한 함수에서만 해당 변수를 사용할 수 있다는 의미인데,  
→ 이런 변수는 **지역 스코프**를 가진다. (지역 변수)
- 프로그램 시작 부분에서 변수를 선언하면 프로그램 전체에서 사용할 수 있다.  
→ 이런 변수는 **전역 스코프**를 가진다. (전역 변수)

```
function sum(a, b) {  
  var result = a + b;  
}  
console.log(result);
```

지역 스코프를 가지는 result 변수

```
var hi = "hello";  
  
function greeting() {  
  console.log(hi);  
}  
  
greeting();
```

전역 스코프를 가지는 hi 변수

# var 변수

함수에서 변수를 선언할 때 변수 이름 앞에 var 예약어를 붙이지 않으면 자바스크립트는 전역 변수로 인식한다.  
실수로 var 예약어 없이 변수를 선언한다면?? → 전역 변수가 되어 엉뚱한 결과가 발생할 수 있다.

var를 넣었을 때와 뺐을 때

어떻게 달라지는지 확인해 보세요

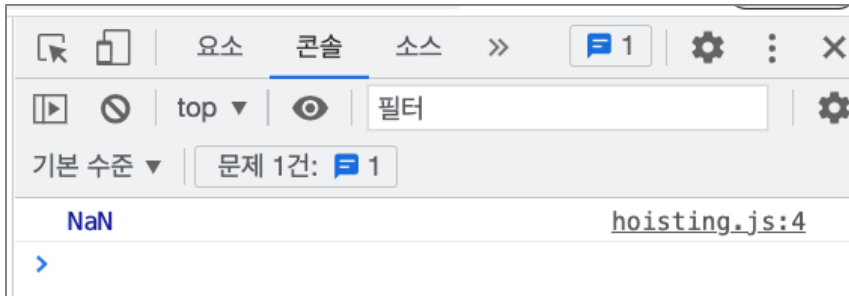
```
function greeting() {  
  hi = "hello";    // 실수로 var를 빠뜨렸다면  
}  
greeting();  
console.log(hi);   // 함수 밖에서 사용할 수 있다.
```

- 여러 함수에서 사용할 변수를 전역 변수로 지정해 놓으면 편리할 수도 있다. **하지만 !!**
- 여러 사람이 공동 작업하는 프로그램일 경우 다른 함수에서 전역 변수를 수정하면 예상하지 못한 결과가 나올 수도 있다.
- 큰 규모의 프로젝트에서는 전역 변수를 사용하는게 위험할 수 있다.

# var 변수와 호이스팅

```
var x = 10;  
var sum = x + y;  
var y = 20;  
console.log(`x : ${x}, y : ${y}, sum : ${sum}`);
```

변수를 선언하기 전에 사용



자바스크립트 해석기가 함수 소스를 훑어보면서 변수를 따로 기억해 두기 때문에, 소스 상에서는 나중에 선언하더라도 내부적으로는 먼저 선언한 것처럼 인식  
→ "호이스팅"

```
var x = 10;  
var y;  
var sum = x + y;  
y = 20;  
...
```

변수 호이스팅은 오류도 발생하지 않으면서 예상 못한 결과를 만든다

→ var 변수를 사용할 때는 호이스팅이 생기지 않도록 주의하자!

# let, const 변수의 스코프 - 블록 스코프

- 자바스크립트에서 이야기하는 블록은 {와 }로 둘러싸인 영역을 가리킨다.
- 블록별로 변수의 유효 범위가 결정되는 것을 블록 스코프(block scope)라고 한다.
- let이나 const를 사용해 만든 변수는 변수가 선언된 블록 안에서만 유효하다. (블록 변수)

```
const factor = 5;

function calc() {
  return num * factor;
}

{
  const num = 10;
  let result = calc();
  console.log(`result : ${result}`)
;
}
```

```
> const factor = 5
< undefined
> function calc() {
  return num * factor;
}
< undefined
> {
  const num = 10;
  let result = calc();
  console.log(`result : ${result}`);
}
✖ ▶ Uncaught ReferenceError: num is not defined VM316:2
   at calc (<anonymous>:2:5)
   at <anonymous>:3:18
>
```

	VM125	VM316 x
1	function calc() {	
2	return num * factor;	✖
3	}	

# 자바스크립트 변수, 이렇게 사용하세요

## var 변수보다 let, const 변수를 사용하자

여러 사람이 함께 진행하는 프로젝트라면

변수를 재선언할 수도 없고 호이스팅도 없는 let이나 const 변수를 사용하는 것이 안전하다.

## 전역 변수는 최소한으로 사용하자

가능하면 전역 변수의 사용을 줄이고, 프로그램에서 값이 변하지 않는다면 const로 선언하는 것이 좋다.

## 객체 선언은 const를 사용하자

객체를 선언할 때에는 프로그램 중에 객체 자체가 바뀌지 않도록 const를 사용해서 선언한다.

객체를 const로 선언해도 객체 안에 있는 프로퍼티는 얼마든지 수정할 수 있다.

---

# 함수 표현식

---

# 함수를 변수에 할당해서 사용하기

```
let sum = function(a, b) {  
  return a + b;  
}  
console.log(`함수 실행 결과 : ${sum(10, 20)}`);
```

익명함수를 사용하는 것은 마치 하나의 값처럼 사용하기 위해서

→ 함수를 변수에 할당할 수도 있고, 다른 함수의 인자로 사용할 수도 있음



# 즉시 실행 함수

지금까지는 함수를 선언한 후 실행 → 같은 함수를 여러번 실행할 수 있음  
한번만 사용하는 함수라면 → 선언과 동시에 실행할 수 있다.

```
(function() {  
    ...  
})();
```

```
(function(매개변수) {  
    .....  
})(인수);
```

인자가 없을 때

```
(function() {  
    let username = prompt('이름을 입력하세요.');
```

```
    alert(`${username}님, 안녕하세요?`);  
})();
```

인자가 있을 때

```
(function(a, b) {  
    let sum = a + b;  
    console.log(`함수 실행 결과 : ${sum}`  
)  
})(100, 200);
```

# 화살표 함수

화살표 함수(애로우 펑션) : ES6 버전부터는 =>을 사용해 함수를 좀 더 간단하게 선언함.

화살표 함수는 함수 표현식을 사용할 경우에만 사용할 수 있음.

( ) => { 함수 내용 }

( 매개변수 ) => { 함수 내용 }

# 화살표 함수 – 매개변수가 없을 때

```
let hi = function() {  
  return `안녕하세요?`;  
}  
hi();
```



```
let hi = () => {return `안녕하세요?`;}  
hi();
```




실행할 명령이 한 줄뿐이라면 중괄호({})를 생략  
한 줄 명령에 return문이 포함되어 있다면 return도 생략



```
let hi = () => `안녕하세요?`;  
hi();
```


# 화살표 함수 – 매개변수가 있을 때

```
let hi = function(user) {  
  console.log(`${user}님, 안녕하세요?`);  
}  
hi("홍길동");
```



```
let hi = user => console.log(`${user}님, 안녕하  
세요?`);  
hi("홍길동");
```

```
let sum = function(a, b) {  
  return a + b;  
}  
sum(10, 20);
```



```
let sum = (a, b) => a + b;  
sum(10, 20);
```

# 콜백 함수

콜백 함수는 다른 함수의 인자로 사용하는 함수

## 함수 이름을 사용해 콜백 함수 실행하기

addEventListener() 함수는 아직 배우지 않았지만, addEventListener() 함수 안에 display() 함수를 인자로 사용한다는 점만 알아두자.

이 때 display 뒤에 괄호가 없다는 점 기억하기!

```
const btn = document.querySelector("button");           // 버튼 요소 가져옴

function display() {
    alert("클릭했습니다.");
}

btn.addEventListener("click", display);                 // 버튼 클릭하면 display
```

함수 실행

# 콜백 함수

## 함수 안에 직접 콜백 함수 작성해서 실행하기

함수 안에서 한번만 실행한다면 함수 안에 직접 콜백 함수를 작성할 수 있다.

```
const btn = document.querySelector("button");    // 버튼 요소 가져옴

btn.addEventListener("click", () => {           // 버튼 클릭하면 alert
실행
    alert("클릭했습니다.");
});
```

---

# 전개 구문

---

# 전개 구문

- 값만 꺼내서 펼쳐주는 구문
- 마침표 3개(...)를 사용해서 표현

```
fruits = ["apple", "banana", "grape"]  
console.log(fruits)
```

```
> console.log(fruits)  
▼ (3) ['apple', 'banana', 'grape'] ⓘ  
  0: "apple"  
  1: "banana"  
  2: "grape"  
  length: 3  
  ► [[Prototype]]: Array(0)
```

```
fruits = ["apple", "banana", "grape"]  
console.log(...fruits)
```

```
> console.log(...fruits)  
apple banana grape  
← undefined
```

← 배열에서 값만 꺼내서 보여줌

문자열이나 배열, 객체처럼 여러 개의 값을 담고 있는 값만 꺼내 사용하려고 할 때 유용함



# 나머지 매개변수

- 마침표 3개를 사용하는 전개 구문은 함수를 선언할 때도 사용
- 함수를 선언하면서 나중에 몇 개의 인 수를 받게 될지 알 수 없는 경우에, 매개변수 자리에 마침표 3개를 사용 → 나머지 매개변수라고 함

(예) 함수를 실행할 때 인수를 몇 개 넣더라도, 함수에서 그걸 모두 더해주는 프로그램을 짜고 싶다면?

```
function addNum(...numbers) {  
  let sum = 0;  
  
  for (let number of numbers)  
    sum += number;  
  
  return sum;  
}
```

```
console.log(addNum(1, 3));  
console.log(addNum(1, 3, 5, 7));
```

# 나머지 매개변수

일부만 변수로 받고 나머지는 한꺼번에 묶어서 받을 수도 있다.

```
function displayFavorites(first, ...fav) {  
  let str = `가장 좋아하는 과일은 "${first}"군요`;   
  return str;  
}  
console.log(displayFavorites("사과", "포도", "토마토"));
```

---

# 타이머 함수

---

# 타이머 함수란

- 특정 시간이 되었을 때 함수를 실행하거나
- 특정 시간 동안 함수를 반복하기 위해서 시간을 재는 함수
- 타이머 함수는 실행할 함수와 시간이 필요하다  
→ 타이머 함수에서 실행할 함수를 인수로 받는다(콜백 함수)

setInterval()

clearInterval()

setTimeout()

타이머 함수의 시간은 밀리초 단위

1s = 1000ms

예) 1초를 지정하려면 1000으로 사용

# setInterval( ) – 일정 시간마다 반복하기

`setInterval(콜백 함수, 시간)`

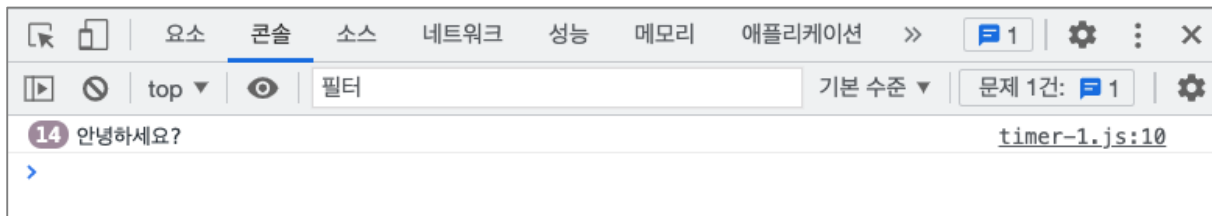
예를 들어, 2초마다 콘솔 창에 인사말을 표시하려면? → `setInterval(인사하는함수, 2000)`

```
function greeting() {  
  console.log("안녕하세요?")  
}  
  
setInterval(greeting, 2000);
```

화살표 함수로  
표현하면



```
setInterval(() => {  
  console.log("안녕하세요?")  
}, 2000);
```



# clearInterval() - 반복 실행 멈추기

- setInterval() 함수는 한 번 실행하면 웹 브라우저를 종료하기 전까지는 계속 실행된다.
- 특정 조건이 되었을 경우 반복 실행을 멈추려면 clearInterval() 사용

`clearInterval(타이머)`

- setInterval()을 사용해서 반복 중인 함수를 '타이머'라고 부름
- 반복 중인 함수를 변수에 저장 → 타이머 변수라고 함

```
let timer = setInterval(() => {  
  console.log("안녕하세요?")  
}, 2000);
```

인삿말을 5번 표시하면 반복을 멈추자!

```
let counter = 0;

let timer = setInterval(() => {      // 타이머 시작
  console.log("안녕하세요?")
  counter++;                          // 인사말 표시 횟수 1 증가
  if (counter === 5)
    clearInterval(timer);            // counter = 5 라면 타이머 종료
}, 2000 );
```

# setTimeout() – 특정 시간 후에 실행하기

지정한 시간이 흐른 후에 괄호 안에 있는 함수(콜백 함수) 실행

```
setTimeout(콜백 함수, 시간)
```

(예) 3초 후에 인사말 표시하기

```
setTimeout(() => {  
  console.log("안녕하세요?")  
}, 3000);
```

웹 브라우저에서 04wtimer-3.html 문서를 열고  
콘솔 창까지 열어둔 상태에서  
[새로 고침] 버튼을 클릭해서,  
콘솔 창에 3초만에 결과가 나오는지 확인하기