JS

함수

(Function)

console.log()

alert()

prompt()

함수
(Function)

typeof()

Number()

String()

함수 정의	특정 행동을 하도록 만드는 명령어 생성
함수 호출	명령어를 이용해, 특정 행동 유도

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.

함수: [추가] 버튼 누르면 목록에 추가하는 기능

함수: 목록을 화면에 표시하는 기능

함수: [완료] 버튼 누르면 취소선 그리는 기능

함수: [삭제] 버튼 누르면 목록서 삭제하는 기능

엄마 해봐~

아빠 해봐~

```
function 해봐 (text) {
    alert(`${text}`)
}

해봐('엄마');
해봐('아빠');
```

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 (매개변수) {
   // 함수 호출 시 실행되는 코드
}
```

함수 실행

```
함수명 (인수);
```

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

```
function errMsg() {
    console.log('에러 발생');
}

errMsg();
```

```
function errMsg (errCode) {
	let newErrCode = errCode || '알 수 없음' ;
	if (errCode) {
		let msg = `에러발생! 에러코드: ${newErrCode}`;
	}
	console.log(msg);
}

errMsg('Login_001');
errMsg('Join_002');
errMsg();
```

```
function errMsg (errCode = '알 수 없음') {
    let msg = `에러코드: ${errCode}`;
    console.log(msg);
}

errMsg('Login_001');
errMsg('Join_002');
errMsg();
```

전역변수

전체에서 접근 가능한 변수

지역변수

함수 내부에서만 접근이 가능한 변수

```
function errMsg (errCode = '알 수 없음') {
    let msg = `에러코드: ${errCode}`;
    console.log(msg);
}

errMsg();
console.log(msg);    // 에러 발생 (정의된 변수가 없음)
```

```
let var_G = '함수의 바깥';

function errMsg (errCode = '알 수 없음') {
    let msg = `에러코드: ${errCode}`;
    console.log(var_G);
}

errMsg();
console.log(var_G);
```

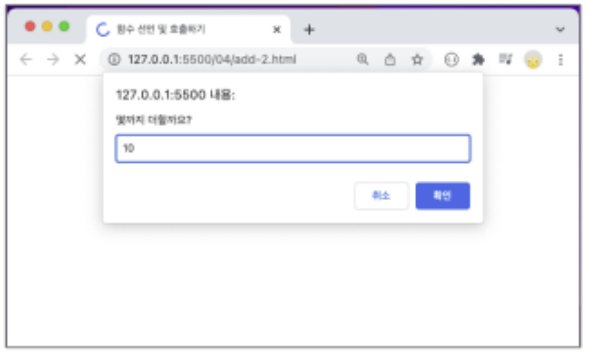
```
let variable = '전역변수';

function variableTest() {
    let variable = '지역변수';
    console.log(`함수를 통해 호출되는 변수는 ${variable}입니다.`);
}

variableTest();
console.log(`바깥에서 호출되는 변수는 ${variable}입니다.`);
```

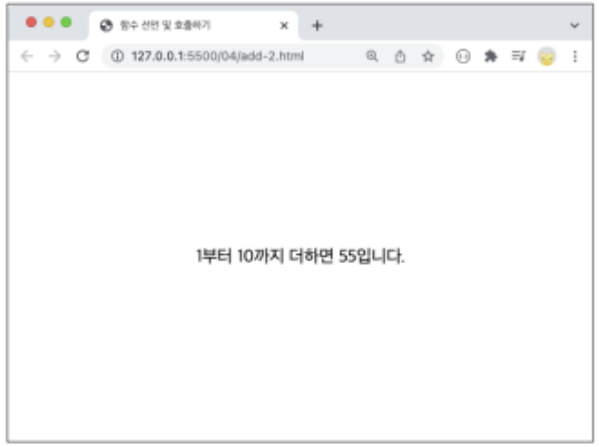
```
let variable = '전역변수';
console.log(`함수 호출 전, 변수는 ${variable}입니다.`);

function variableTest() {
    variable = '변경된 전역변수';
    console.log(`함수를 통해 호출되는 변수는 ${variable}입니다.`);
}

variableTest();

console.log(`함수 호출 후, 변수는 ${variable}입니다.`);
```

return

- 함수 안에서 실행하고 그 결과를 함수 밖에서 받아 처리해야 할 경우도 많다.
- 함수의 실행 결과를 함수를 실행한 시점으로 넘겨주는 것을 '결괏값을 반환한다return'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)}입니다.`);
```





```
function rectangle ( width, height = null ) {
      if (height === null) {
            return width * width;
      } else {
            return width * height;
      }
}

let squareArea = rectangle(3);
console.log(squareArea);

let rectangleArea = rectangle(3, 4);
console.log(rectangleArea);
```

```
function errMsg (errCode = '알 수 없음') {
    let msg = `에러코드: ${errCode}`;
    alert(msg);
}

let result = errMsg();

console.log(result);
```

```
function errMsg (errCode = '알 수 없음') {
    let msg = `에러코드: ${errCode}`;
    alert(msg);
    return;
}

let result = errMsg();

console.log(result);
```

```
function errMsg (errCode = '알 수 없음') {
    let msg = `에러코드: ${errCode}`;
    console.log(msg);
    return;
    console.log('함수가 종료되었기 때문에 절대 실행되지 않는 내용입니다.');
}

let result = errMsg();
```

함수 표현식

```
let errMsg = function (errCode = '알 수 없음') {
    let msg = `에러코드: ${errCode}`;
    console.log(msg);
}

errMsg();
```

```
function errMsg (errCode = '알 수 없음') {
    let msg = `에러코드: ${errCode}`;
    console.log(msg);
}

errMsg();
```

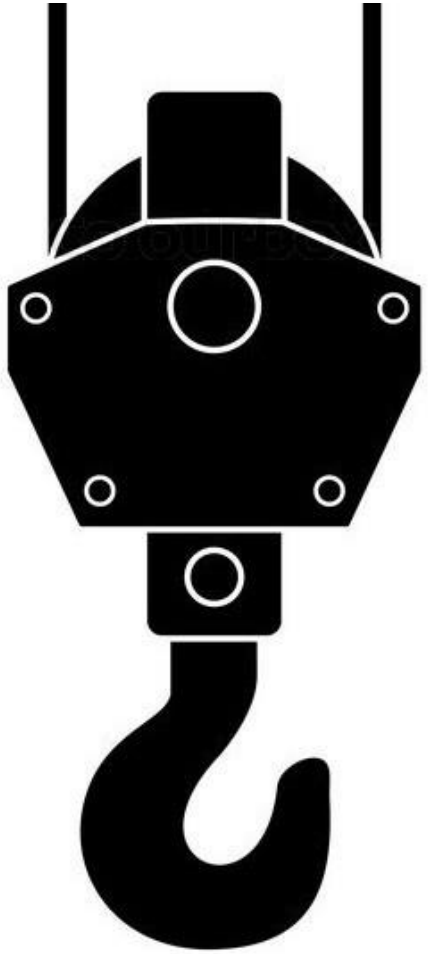
```
let errMsg = function (errCode = '알 수 없음') {
    let msg = `에러코드: ${errCode}`;
    console.log(msg);
}

errMsg();
```

```
console.log(num);

let num = '콘솔에 찍힐까?';
```

```
errMsg();

function errMsg (errCode = '알 수 없음') {
    let msg = `에러코드: ${errCode}`;
    console.log(msg);
}
```

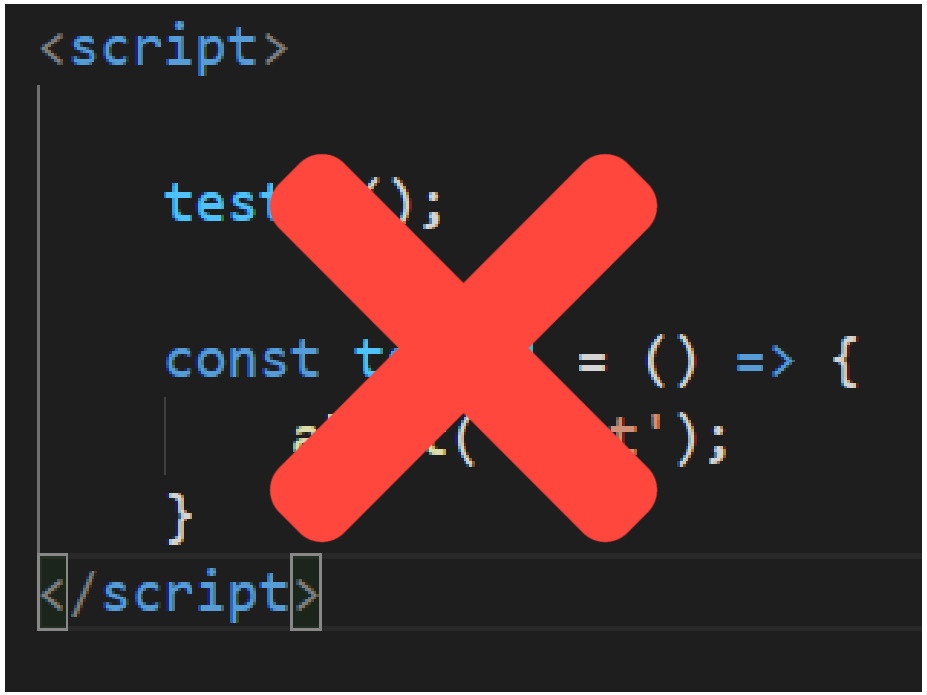
```
errMsg();

let errMsg = function (errCode = '알 수 없음') {
    let msg = `에러코드: ${errCode}`;
    console.log(msg);
}
```


JavaScript



JAVASCRIPT HOISTING

```
1  <script>
2
3      test01();
4
5      function test01() {
6          alert('test');
7      }
8  </script>
```



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 함수는
함수표현식에서 function이 사라지는 대신
소괄호와 중괄호 사이에 화살표가 발생하는 함수이다.

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, 안녕하세요?`);
hi("홍길동");
```

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

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

```
let rectangle = function ( w, h = null ) {
      if (h === null) {
            return w * w;
      } else {
            return w * h;
      }
}
```

```
let rectangle = ( w, h = null ) => {
   if (h === null) {
      return w * w;
   } else {
      return w * h;
   }
}
```

```
let rectangle = ( width, height ) => {
    return width * height;
}
```

```
let rectangle = ( width, height ) => width * height;
```

```
let errMsg = (errCode) => {
        let newErrCode = errCode || '알 수 없음' ;
        if (errCode) {
                let msg = `에러코드: ${newErrCode}`;
        }
        console.log(msg);
}
```

```
let errMsg = errCode => {
        let newErrCode = errCode || '알 수 없음' ;
        if (errCode) {
                let msg = `에러코드: ${newErrCode}`;
        }
        console.log(msg);
}
```

```
let errMsg = () => console.log('에러 발생');
```

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 bttn = document.querySelector("button");      // 버튼 요소 가져옴

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

bttn.addEventListener("click", display);          // 버튼 클릭하면 display 함수 실행
```

콜백 함수

함수 안에 직접 콜백 함수 작성해서 실행하기

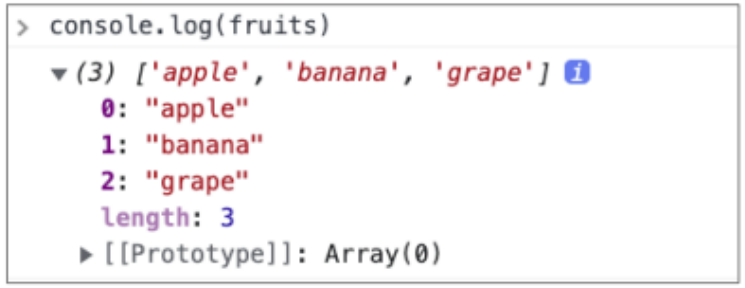
함수 안에서 한번만 실행한다면 함수 안에 직접 콜백 함수를 작성할 수 있다.

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

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

전개 구문

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

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




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

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 한꺼번에서 묶어서 받을 수도 있다.

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

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)

타이머 함수의 시간은 밀리초 단위

1s = 1000ms

예) 1초를 지정하려면 1000으로 사용

setInterval()

clearInterval()

setTimeout()

setInterval( ) – 일정 시간마다 반복하기

setInterval(*콜백 함수, 시간*)

예를 들어, 2초마다 콘솔 창에 인삿말을 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 인삿말 표시하기

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

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