

혼자 공부하는 자바스크립트

Chapter 05 함수





Contents

- CHAPTER 05: 함수

SECTION 5-1 함수의 기본 형태

SECTION 5-2 함수 고급



CHAPTER 05 함수

다양한 형태의 함수를 만들기과 매개변수를 다루는 방법 이해

SECTION 5-1 함수의 기본 형태(1)

- 익명 함수

- 함수는 코드의 집합을 나타내는 자료형

```
function () { }
```

- 함수 표현식 - Function Expressions

- 익명 함수를 이용하여 함수를 표현

```
let funcExpression = function () {  
    return 'A function expression';  
}  
funcExpression();
```

- 선언적 함수

- 선언적 함수는 이름을 붙여 생성

```
function 함수() {  
}
```

- 함수 선언식 - Function Declarations

- 일반 프로그래밍의 함수 선언방식

```
function funcDeclarations() {  
    return 'A function declaration';  
}  
funcDeclarations();
```

SECTION 5-1 함수의 기본 형태(2)

- 익명 함수

- 익명 함수 선언하기 (소스 코드 5-1-1.html)

```
<script>
// 변수를 생성합니다.
const func1 = function () {
  console.log('함수 내부의 코드입니다 ... 1');
  console.log('함수 내부의 코드입니다 ... 2');
  console.log("");
};
```

```
// 함수를 호출합니다.
```

```
func1();
```

→ 우리가 만든 함수도 기존의 alert(), prompt() 함수처럼 호출할 수 있음

```
// 출력합니다.
```

```
console.log(typeof func1);
```

→ 함수의 자료형을 확인

```
console.log(func1);
```

→ 함수 자체도 단순한 자료이므로 출력 가능

```
</script>
```

SECTION 5-1 함수의 기본 형태(3)

- 선언적 함수
 - 선언적 함수 선언하기 (소스 코드 5-1-2.html)

```
<script>
// 함수를 생성합니다.
function func2() {
  console.log('함수 내부의 코드입니다 ... 1');
  console.log('함수 내부의 코드입니다 ... 2');
  console.log("");
}

// 함수를 호출합니다.
func2();

// 출력합니다.
console.log(typeof func2);
console.log(func2);
</script>
```

SECTION 5-1 함수의 기본 형태(4)

- 매개변수(Parameter)와 리턴값

- prompt() 함수의 매개변수와 리턴값

```
function prompt(message?:string,_default?:string): string
```

```
Prompt()
```

- 사용자 정의 함수의 매개변수와 리턴값

```
function 함수():void
```

```
함수()
```

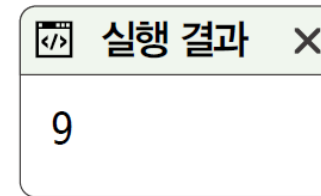
- 매개변수와 리턴값을 갖는 함수

```
function 함수(매개변수, 매개변수, 매개변수) {  
  문장  
  문장  
  return 리턴값  
}
```

SECTION 5-1 함수의 기본 형태(5)

- 매개변수와 리턴값
 - 기본 형태의 함수 만들기 (소스 코드 5-1-3.html)

```
<script>  
  // 함수를 선언합니다.  
  function f(x) {  
    return x * x;  
  }  
  
  // 함수를 호출합니다.  
  console.log(f(3));  
</script>
```



SECTION 5-1 함수의 기본 형태(6)

- 기본적인 함수 예제 실습
 - A부터 B까지 더하는 함수 만들기
 - A부터 B까지 더하는 함수 만들기
 - a부터 b까지 더하는 함수 (소스 코드 5-1-5.html)

```
<script>
function sumAll(a, b) {
  let output = 0;

  for (let i = a; i <= b; i++) {
    output += i;
  }
  return output;
}

const sum = sumAll(1, 100);
console.log(`1부터 100까지의 합: ${sum}`);
</script>
```

SECTION 5-1 함수의 기본 형태(7)

◦ 나머지 매개변수

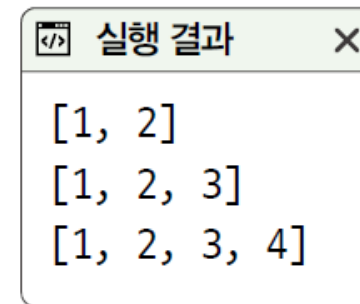
- 가변 매개변수 함수: 호출할 때 매개변수의 개수가 고정적이지 않은 함수
- 자바스크립트에서 이러한 함수를 구현할 때 **나머지 매개변수**(rest parameter, ES6)라는 특이한 형태의 문법을 사용

```
function 함수 이름(...나머지 매개변수) {}
```

- 나머지 매개변수를 사용한 배열 만들기 (소스 코드 5-1-7.html)

```
<script>
  function sample(...items) {
    console.log(items);
  }

  sample(1, 2);
  sample(1, 2, 3);
  sample(1, 2, 3, 4);
</script>
```



```
실행 결과
[1, 2]
[1, 2, 3]
[1, 2, 3, 4]
```


SECTION 5-1 함수의 기본 형태(8)

- 나머지 매개변수

- 나머지 매개변수를 사용한 min() 함수 (소스 코드 5-1-8.html)

```
<script>
function min(...items) {
  let output = items[0];
  for (const item of items) {
    if (output > item) {
      output = item;
    }
  }
  return output;
}

// 함수 호출하기
console.log('min(52, 273, 32, 103, 275, 24, 57)');
console.log(`=${min(52, 273, 32, 103, 275, 24, 57)}`);
</script>
```

 실행 결과

min(52, 273, 32, 103, 275, 24, 57)
= 24

SECTION 5-1 함수의 기본 형태(9)

◦ 나머지 매개변수

- 나머지 매개변수와 일반 매개변수 조합하기

```
function 함수 이름(매개변수, 매개변수, ...나머지 매개변수) {}
```

- 나머지 매개변수와 일반 매개변수를 갖는 함수 (소스 코드 5-1-9.html)

```
<script>
  function sample(a, b, ...c) {
    console.log(a, b, c);
    console.log(typeof c, Array.isArray(c));
  }

  sample(1, 2);
  sample(1, 2, 3);
  sample(1, 2, 3, 4);
</script>
```

어떤 자료가 배열인지 확인할 때는 `Array.isArray()` 메소드를 사용 (`typeof` 연산자로만 배열을 확인할 수 없음)

DEBUG CONSOLE

PROBLEMS

```
> 1 2 (0) []
  object true
> 1 2 (1) [3]
  object true
> 1 2 (2) [3, 4]
  object true
```

SECTION 5-1 함수의 기본 형태(10)

◦ 나머지 매개변수

- 전개 연산자: 배열을 전개해서 함수의 매개변수로 전달
- 전개 연산자의 활용 (소스 코드 5-1-11.html)

```
<script>
// 단순히 매개변수를 모두 출력하는 함수
function sample(...items) {
  console.log(items);
}

// 전개 연산자 사용 여부 비교하기
const array = [1, 2, 3, 4];

console.log('# 전개 연산자를 사용하지 않은 경우');
sample(array);
console.log('# 전개 연산자를 사용한 경우');
sample(...array);
</script>
```

실행 결과

전개 연산자를 사용하지 않은 경우

[Array(4)] → 4개의 요소가 있는 배열이 들어옴

전개 연산자를 사용한 경우

[1, 2, 3, 4] → 숫자가 하나하나 들어옴

SECTION 5-1 함수의 기본 형태(11)

◦ 기본 매개변수

- 여러 번 반복 입력되는 매개변수에 기본값을 지정하여 사용
 - 기본 매개변수는 오른쪽 매개변수에 사용

함수 이름(매개변수, 매개변수=기본값, 매개변수=기본값)

- 매개변수로 시급과 시간을 입력받아 급여를 계산하는 함수 연습
 - 함수 이름: earnings
 - 매개변수: name(이름), wage(시급), hours(시간)
 - 함수의 역할: 이름, 시급, 시간을 출력하고, 시급과 시간을 곱한 최종 급여 출력
 - 만약 wage와 hours를 입력하지 않고 실행하면 wage에 최저 임금이 들어가고, hours에 법정근로시간 1주일 40시간이 기본 매개변수로 입력

SECTION 5-1 함수의 기본 형태(18)

- 기본 매개변수
 - 기본 매개변수의 활용 (소스 코드 5-1-12.html)

```
<script>
function earnings(name, wage = 9620, hours = 40) {
  console.log(`# ${name} 님의 급여 정보`);
  console.log(`- 시급: ${wage}원`);
  console.log(`- 근무 시간: ${hours}시간`);
  console.log(`- 급여: ${wage * hours}원\n`);
}

// 최저 임금으로 최대한 일하는 경우
earnings('구름');

// 시급 1만원으로 최대한 일하는 경우
earnings('별', 10000);

// 시급 1만원으로 52시간 일한 경우
earnings('인성', 10000, 52);
</script>
```

DEBUG CONSOLE PROBLEMS

```
# 구름 님의 급여 정보
- 시급: 9620원
- 근무 시간: 40시간
- 급여: 384800원

# 별 님의 급여 정보
- 시급: 10000원
- 근무 시간: 40시간
- 급여: 400000원

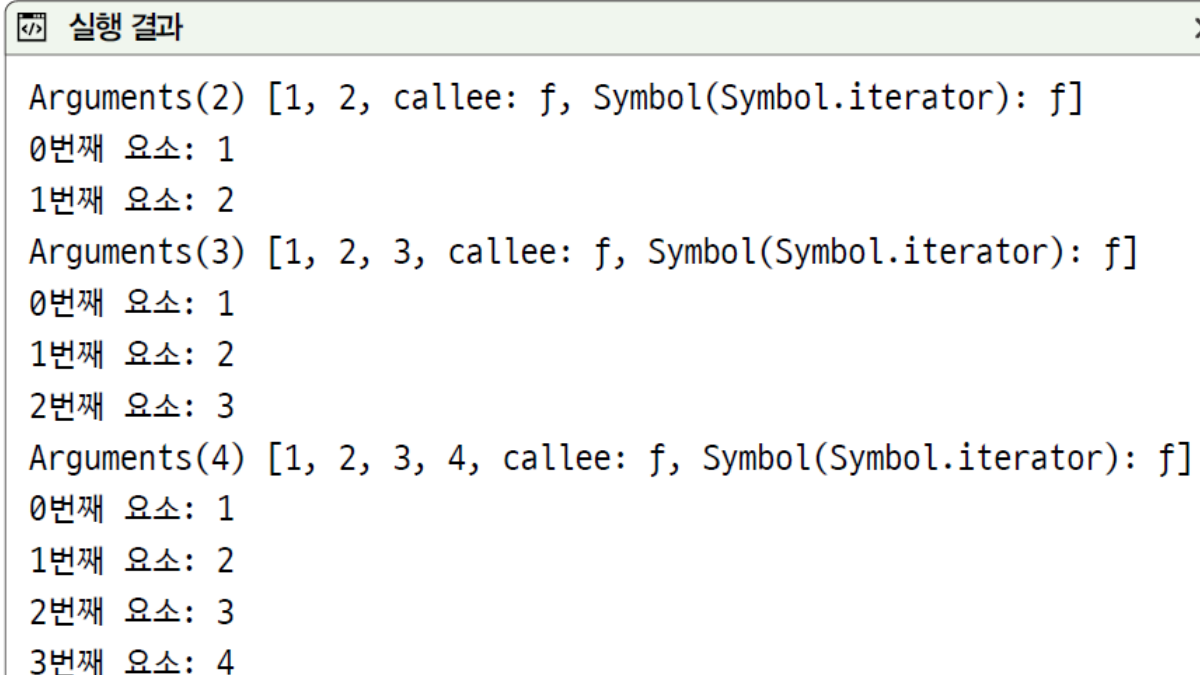
# 인성 님의 급여 정보
- 시급: 10000원
- 근무 시간: 52시간
- 급여: 520000원
```

[좀 더 알아보기①] 구 버전 자바스크립트에서 가변 매개변수 함수 구현하기

- 구 버전의 자바스크립트에서 가변 매개변수 함수를 구현할 때는 배열 내부에서 사용할 수 있는 특수한 변수인 arguments를 활용
- arguments를 사용한 가변 매개변수 함수 (소스 코드 5-1-14.html)

```
<script>
function sample() {
  console.log(arguments);
  for (let i = 0; i < arguments.length; i++) {
    console.log(`${i}번째 요소: ${arguments[i]}`);
  }
}

sample(1, 2);
sample(1, 2, 3);
sample(1, 2, 3, 4);
</script>
```



```
실행 결과
Arguments(2) [1, 2, callee: f, Symbol(Symbol.iterator): f]
0번째 요소: 1
1번째 요소: 2
Arguments(3) [1, 2, 3, callee: f, Symbol(Symbol.iterator): f]
0번째 요소: 1
1번째 요소: 2
2번째 요소: 3
Arguments(4) [1, 2, 3, 4, callee: f, Symbol(Symbol.iterator): f]
0번째 요소: 1
1번째 요소: 2
2번째 요소: 3
3번째 요소: 4
```


[좀 더 알아보기②] 구 버전 자바스크립트에서 기본 매개변수 구현하기

- 함수의 매개변수에 바로 값을 입력하는 기본 매개변수는 최신 자바스크립트에서 추가된 기능
- 구 버전의 자바스크립트에서는 일반적으로 다음과 같은 코드를 사용해서 기본 매개변수를 구현

```
function earnings (wage, hours) {  
  wage = typeof(wage) !== undefined ? wage : 8590;  
  hours = typeof(hours) !== undefined ? hours : 52;  
  return wage * hours;  
}
```

- 매개변수로 들어오는 값이 false 또는 false로 변환되는 값(0, 빈 문자열 등)이 아니라는 게 확실하다면 다음과 같이 짧은 조건문을 사용해서 기본 매개변수를 구현

```
function earnings (wage, hours) {  
  wage = wage || 8590;  
  hours = hours || 52;  
  return wage * hours;  
}
```

[요점 정리]

◦ 7가지 키워드로 정리하는 핵심 포인트



- **익명 함수**란 이름이 없는 함수로 `function () {}` 형태로 만들
- **선언적 함수**란 이름이 있는 함수로 `function 함수 이름 () {}` 형태로 만들
- 함수의 괄호 안에 넣는 변수를 **매개변수**라고 합니다. 매개변수를 통해 함수는 외부의 정보를 입력 받을 수 있음
- 함수의 최종적인 결과를 **리턴값**이라고 합니다. 함수 내부에 `return` 키워드를 입력하고 뒤에 값을 넣어서 생성
- **가변 매개변수** 함수란 매개변수의 개수가 고정되어 있지 않은 함수를 의미. 나머지 매개변수(...)를 활용해서 만들
- **전개 연산자**란 배열을 함수의 매개변수로서 전개하고 싶을 때 사용
- **기본 매개변수**란 매개변수에 기본값이 들어가게 하고 싶을 때 사용하는 매개변수

SECTION 5-2 함수 고급(1)

콜백 함수

- 자바스크립트는 함수도 하나의 자료형이므로 매개변수로 전달할 수 있는데, 이렇게 매개변수로 전달하는 함수를 콜백(callback) 함수
- 콜백 함수(1): 선언적 함수 사용하기 (실습)(소스 코드 5-2-1.html)

```
01 <script>
02 // 함수를 선언합니다.
03 function callThreeTimes (callback) {
04   for (let i = 0; i < 3; i++) {
05     callback(i); → callback이라는 매개변수는 함수이므로 호출할 수 있음
06   }
07 }
08
09 function print (i) {
10   console.log(`${i}번째 함수 호출`);
11 }
12
13 // 함수를 호출합니다.
14 callThreeTimes(print);
15 </script>
```

 실행 결과 

0번째	함수	호출
1번째	함수	호출
2번째	함수	호출

SECTION 5-2 함수 고급(2)

콜백 함수

- 콜백 함수(2): 익명 함수 사용하기 (소스 코드 5-2-2.html)

```
01 <script>
02 // 함수를 선언합니다.
03 function callThreeTimes(callback) {
04   for (let i = 0; i < 3; i++) {
05     callback(i);
06   }
07 }
08
09 // 함수를 호출합니다.
10 callThreeTimes(function (i) {
11   console.log(`${i}번째 함수 호출`);
12 });
13 </script>
```

→ 익명 함수 사용하기

실행 결과

0번째	함수	호출
1번째	함수	호출
2번째	함수	호출

SECTION 5-2 함수 고급(3)

콜백 함수

- 콜백 함수를 활용하는 함수: `forEach()`
- `forEach()` 메소드는 배열이 갖고 있는 함수(메소드)로써 단순히 배열 내부의 요소를 사용해서 콜백 함수를 호출

```
function (value, index, array) { }
```

- 배열의 `forEach()` 메소드 (소스 코드 5-2-3.html)

```
01 <script>
02  const numbers = [273, 52, 103, 32, 57];
03
04  numbers.forEach(function (value, index, array) {
05    console.log(`${index}번째 요소 : ${value}`);
06  })
07 </script>
```

매개변수로 `value, index, array`를 갖는 콜백 함수를 사용

실행 결과

0번째 요소	: 273
1번째 요소	: 52
2번째 요소	: 103
3번째 요소	: 32
4번째 요소	: 57

SECTION 5-2 함수 고급(4)

콜백 함수

- 콜백 함수를 활용하는 함수: map()
- map() 메소드는 콜백 함수에서 리턴한 값들을 기반으로 새로운 배열을 만드는 함수
- 배열의 map() 메소드 (소스 코드 5-2-4.html)

```
01 <script>
02 // 배열을 선언합니다.
03 let numbers = [273, 52, 103, 32, 57];
04
05 // 배열의 모든 값을 제공합니다.
06 numbers = numbers.map(function (value, index, array) {
07     return value * value;
08 });
09
10 // 출력합니다.
11 numbers.forEach(console.log);
12 </script>
```

매개변수로 value, index, array를
갖는 콜백 함수를 사용

매개변수로 console.log 메소드
자체를 넘김

실행 결과

74529	0	Array(5)
2704	1	Array(5)
10609	2	Array(5)
1024	3	Array(5)
3249	4	Array(5)

SECTION 5-2 함수 고급(5)

- 콜백 함수

- 원하는 매개변수만 받기 (소스 코드 5-2-4-1.html)

```
<script>
// 배열을 선언합니다.
let numbers = [273, 52, 103, 32, 57];
// 배열의 모든 값을 제공합니다.
numbers = numbers.map(function (value) {
    return value * value;
})
// 출력합니다.
numbers.forEach(console.log);
</script>
```

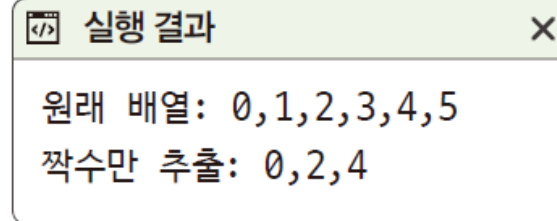
함수 내부에서 value만 사용하므로
value만 매개변수로 넣음

SECTION 5-2 함수 고급(6)

콜백 함수

- 콜백 함수를 활용하는 함수: filter()
- filter() 메소드는 콜백 함수에서 리턴하는 값이 true인 것들만 모아서 새로운 배열을 만드는 함수
- 배열의 filter() 메소드 (소스 코드 5-2-5.html)

```
01 <script>
02  const numbers = [0, 1, 2, 3, 4, 5];
03  const evenNumbers = numbers.filter(function (value) {
04    return value % 2 === 0;
05  })
06
07  console.log(`원래 배열: ${numbers}`);
08  console.log(`짝수만 추출: ${evenNumbers}`);
09 </script>
```



실행 결과

원래 배열: 0,1,2,3,4,5
짝수만 추출: 0,2,4

SECTION 5-2 함수 고급(7)

- 화살표 함수(arrow function)

- 화살표 함수는 function 키워드 대신 화살표(=>)를 사용하며, 일반 함수 표현식보다 단순하고 간결한 문법 사용

```
let func = (param1, param2, ...paramN) => expression
```

```
let func = function(param1, param2, ...paramN) {  
  return expression;  
};
```

- 매개변수가 하나인 경우 : 괄호 생략 가능

```
(param1) => expression  
param1 => expression
```

- 매개변수가 없는 경우 : 괄호 필요

```
() => expression
```

SECTION 5-2 함수 고급(7)

- 화살표 함수(arrow function) 예제

```
<script>
let elements = ['Hydrogen', 'Helium', 'Lithium', 'Beryllium'];
let arr1 = elements.map(function(element) {
  return element.length;
});
console.log(arr1); // [8, 6, 7, 9]

// 위의 일반적인 함수 표현은 아래 화살표 함수로 쓸 수 있다.
let arr2 = elements.map((element) => {
  return element.length;
});
console.log(arr2); // [8, 6, 7, 9]

// 파라미터가 하나만 있을 때는 주변 괄호를 생략할 수 있다.
let arr3 = elements.map(element => {
  return element.length;
});
console.log(arr3); // [8, 6, 7, 9]

// 화살표 함수의 유일한 문장이 'return'일 때 'return'과 중괄호({})를 생략할 수 있다.
let arr4 = elements.map(element => element.length);
console.log(arr4); // [8, 6, 7, 9]
</script>
```

SECTION 5-2 함수 고급(7)

- 화살표 함수(arrow function)
 - 배열의 메소드와 화살표 함수 (소스 코드 5-2-6.html)

```
01 <script>
02 // 배열을 선언합니다.
03 let numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9];
04
05 // 배열의 메소드를 연속적으로 사용합니다.
06 numbers
07   .filter((value) => value % 2 === 0 )
08   .map((value) => value * value)
09   .forEach((value) => {
10     console.log(value);
11   })
12 </script>
```

메소드 체이닝

실행 결과

0
4
16
36
64

SECTION 5-2 함수 고급(8)

타이머 함수

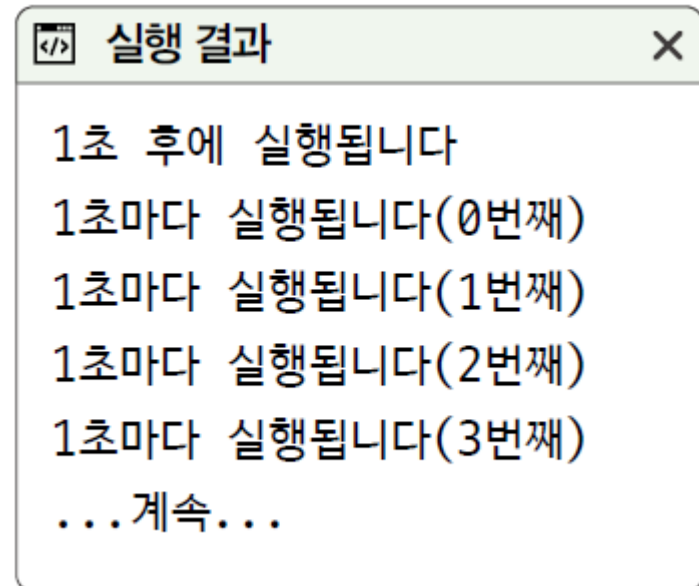
- 특정 시간마다 또는 특정 시간 이후에 콜백 함수를 호출할 수 있는 타이머(timer) 함수

함수 이름	설명
setTimeout(함수, 시간)	특정 시간 후에 함수를 한 번 호출
setInterval(함수, 시간)	특정 시간마다 함수를 호출

- 타이머 걸기 (소스 코드 5-2-7.html)

```
01 <script>
02   setTimeout(() => {
03     console.log('1초 후에 실행됩니다');
04   }, 1 * 1000);
05
06   let count = 0
07   setInterval(() => {
08     console.log(`1초마다 실행됩니다(${count}번째)`);
09     count++;
10   }, 1 * 1000);
11 </script>
```

웹 브라우저를 강제 종료해 멈춤 →



SECTION 5-2 함수 고급(9)

◦ 타이머 함수

- 타이머를 종료하고 싶을 때는 clearTimeout() 함수와 clearInterval() 함수를 사용

함수 이름	설명
clearTimeout(타이머_ID)	setTimeout() 함수로 설정한 타이머를 제거
clearInterval(타이머_ID)	setInterval() 함수로 설정한 타이머를 제거

- 타이머 취소하기 (소스 코드 5-2-8.html)

```
01 <script>
02 let id;
03 let count = 0;
04 id = setInterval(() => {
05   console.log(`1초마다 실행됩니다(${count}번째)`);
06   count++;
07 }, 1 * 1000);
08
09 setTimeout(() => {
10   console.log('타이머를 종료합니다.~');
11   clearInterval(id);
12 }, 5 * 1000);
13 </script>
```

실행 결과

```
1초마다 실행됩니다(0번째)
1초마다 실행됩니다(1번째)
1초마다 실행됩니다(2번째)
1초마다 실행됩니다(3번째)
1초마다 실행됩니다(4번째)
타이머를 종료합니다.
```

[좀 더 알아보기①] 즉시 호출 함수

- 함수 즉시 호출하기

```
(function () {} )()
```

- 이름 충돌 문제 발생 (소스 코드 5-2-9.html)

```
01 <!-- 다른 곳에서 가져온 자바스크립트 코드 -->
02 <script>
03  let pi = 3.14;
04  console.log(`파이 값은 ${pi}입니다.`);
05 </script>
06
07 <!-- 내가 만든 자바스크립트 코드 -->
08 <script>
09  let pi = 3.141592;
10  console.log(`파이 값은 ${pi}입니다.`);
11 </script>
```

실행 결과

파이 값은 3.14입니다.

⊗ Uncaught SyntaxError: Identifier 'pi' has already been declared

식별자가 이미 사용되고 있다는 오류를 발생하면서
<!-- 내가 만든 자바스크립트 코드 -->라는 부분이 실행되지 않음

[좀 더 알아보기①] 즉시 호출 함수

- 블록과 함수 블록을 사용해 이름 충돌 문제 해결하기 (소스 코드 5-2-10.html)

```
01 <!-- 다른 곳에서 가져온 자바스크립트 코드 -->
```

```
02 <script>
```

```
03 let pi = 3.14;
```

```
04 console.log(`파이 값은 ${pi}입니다.`);
```

```
05
```

```
06 // 블록을 사용한 스코프 생성
```

```
07 {
```

```
08   let pi = 3.141592;
```

```
09   console.log(`파이 값은 ${pi}입니다.`);
```

```
10 }
```

```
11 console.log(`파이 값은 ${pi}입니다.`);
```

```
12
```

```
13 // 함수 블록을 사용한 스코프 생성
```

```
14 function sample() {
```

```
15   let pi = 3.141592;
```

```
16   console.log(`파이 값은 ${pi}입니다.`);
```

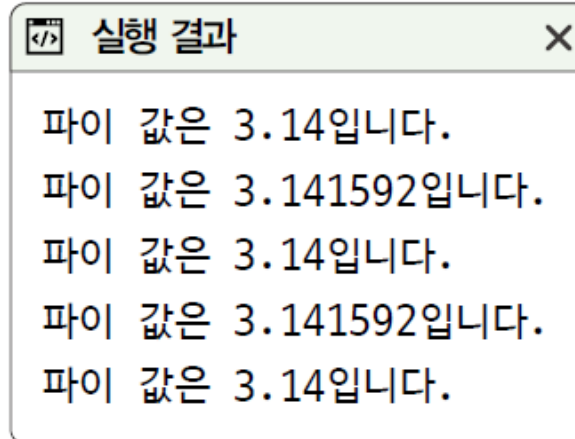
```
17 }
```

```
18 sample();
```

```
19 console.log(`파이 값은 ${pi}입니다.`);
```

```
20 </script>
```

다른 블록에 속하므로 변수 이름 충돌이
발생하지 않음



[좀 더 알아보기②] 즉시 호출 함수 문제 해결하기

- 블록과 함수 블록을 사용해 이름 충돌 문제 해결하기 (소스 코드 5-2-10.html)
 - 블록을 사용하는 방법과 함수 블록을 사용해 변수 충돌을 막는 방법 모두 최신 자바스크립트를 지원하는 웹 브라우저에서는 사용할 수 있음
 - 하지만 구 버전의 자바스크립트에서 변수를 선언할 때 사용하던 var 키워드는 함수 블록을 사용하는 경우에만 변수 충돌을 막을 수 있음
- 즉시 호출 함수를 사용한 문제 해결 (소스 코드 5-2-11.html)

```
01 <!-- 다른 곳에서 가져온 자바스크립트 코드 -->
```

```
02 <script>
```

```
03 let pi = 3.14
```

```
04 console.log(`파이 값은 ${pi}입니다.`)
```

```
05 </script>
```

```
06 <!-- 내가 만든 자바스크립트 코드 -->
```

```
07 <script>
```

```
08 (function () {
```

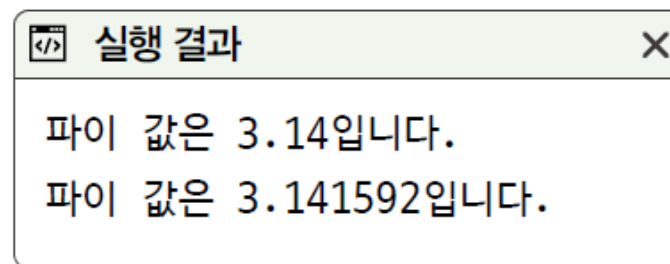
```
09   let pi = 3.141592
```

```
10   console.log(`파이 값은 ${pi}입니다.`)
```

```
11 })()
```

```
12 </script>
```

즉시 호출 함수를 사용해
변수 이름 충돌 문제를 해결



[좀 더 알아보기③] 엄격 모드

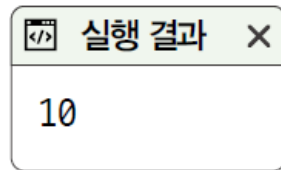
- 엄격 모드

- 여러 자바스크립트 코드를 보면 블록의 가장 위쪽에 **'use strict'**라는 문자열
- 이는 엄격 모드(strict mode) 기능으로 자바스크립트는 이러한 문자열을 읽어들이는 순간부터 코드를 엄격하게 검사

```
<script>  
  'use strict'  
  문장  
  문장  
</script>
```

- 선언없이 변수 사용 (소스 코드 5-2.12.html)

```
01 <script>  
02  data = 10;  
03  console.log(data);  
04 </script>
```

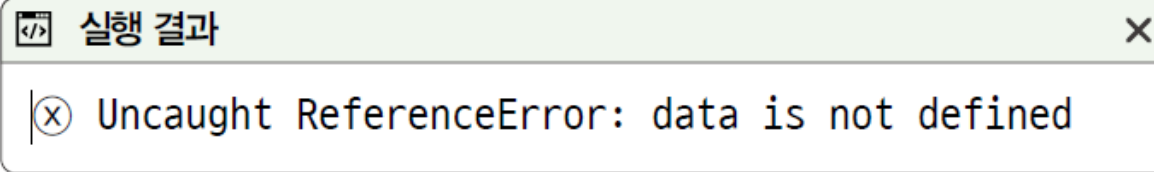


※ 엄격 모드에서는 이러한 코드를 사용할 수 없음
변수를 let 키워드 등으로 선언하지 않았는데 사용했다고 곧바로 오류가 발생

[좀 더 알아보기③] 엄격 모드

- 엄격 모드에서 선언 없이 변수 사용 (소스 코드 5-2-13.html)

```
01 <script>  
02 'use strict'  
03 data = 10;  
04 console.log(data);  
05 </script>
```



- 모질라 엄격 모드 문서 참조

URL https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Strict_mode

엄격 모드는 평범한 JavaScript 시멘틱스에 몇가지 변경이 일어나게 합니다.

1. 기존에는 조용히 무시되던 에러들을 throwing합니다.
2. JavaScript 엔진의 최적화 작업을 어렵게 만드는 실수들을 바로잡습니다.
가끔씩 엄격 모드의 코드는 비-엄격 모드의 동일한 코드보다 더 빨리 작동하도록 만들어집니다.
3. 엄격 모드는 ECMAScript의 차기 버전들에서 정의 될 문법을 금지합니다.

[좀 더 알아보기④] 익명 함수와 선언적 함수의 차이

- 익명 함수의 사용
 - 익명 함수는 순차적인 코드 실행에서 코드가 해당 줄을 읽을 때 생성됨
- 익명 함수 호출 (소스 코드 5-2-14.html)

```
<script>
// 익명 함수를 호출합니다.
funcExpression();

// 익명 함수를 생성합니다.
let funcExpression = function () {
  console.log('익명 함수입니다.');
```

DEBUG CONSOLE PROBLEMS OUTPUT TERMI

Uncaught ReferenceError ReferenceError:

[좀 더 알아보기④] 익명 함수와 선언적 함수의 차이

- 선언적 함수의 사용
 - 선언적 함수는 순차적인 코드 실행이 일어나기 전에 생성됨
- 선언적 함수 호출 (소스 코드 5-2-15.html)

```
<script>
// 선언적 함수를 호출합니다.
funcDeclaration();

function funcDeclaration() {
  console.log('선언적 함수입니다.');
```

- 호이스팅(hoisting) : 인터프리터가 변수와 함수의 메모리 공간을 선언 전에 미리 할당하는 것
선언을 코드의 최상단으로 끌어올림

[요점 정리]

- 4가지 키워드로 정리하는 핵심 포인트
 - 콜백 함수란 매개변수로 전달하는 함수를 의미
 - 화살표 함수란 익명 함수를 간단하게 사용하기 위한 목적으로 만들어진 함수 생성 문법 `() => {}` 형태로 함수를 만들고, 리턴값만을 가지는 함수라면 `() => 값` 형태로 사용할 수 있음
 - 즉시 호출 함수란 변수의 이름 충돌을 막기 위해서 코드를 안전하게 사용하는 방법
 - 자바스크립트의 문법 오류를 더 발생시키는 **엄격 모드**는 실수를 줄일 수 있는 방법
'use strict'라는 문자열을 블록 가장 위쪽에 배치해서 사용