

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 () { }
```

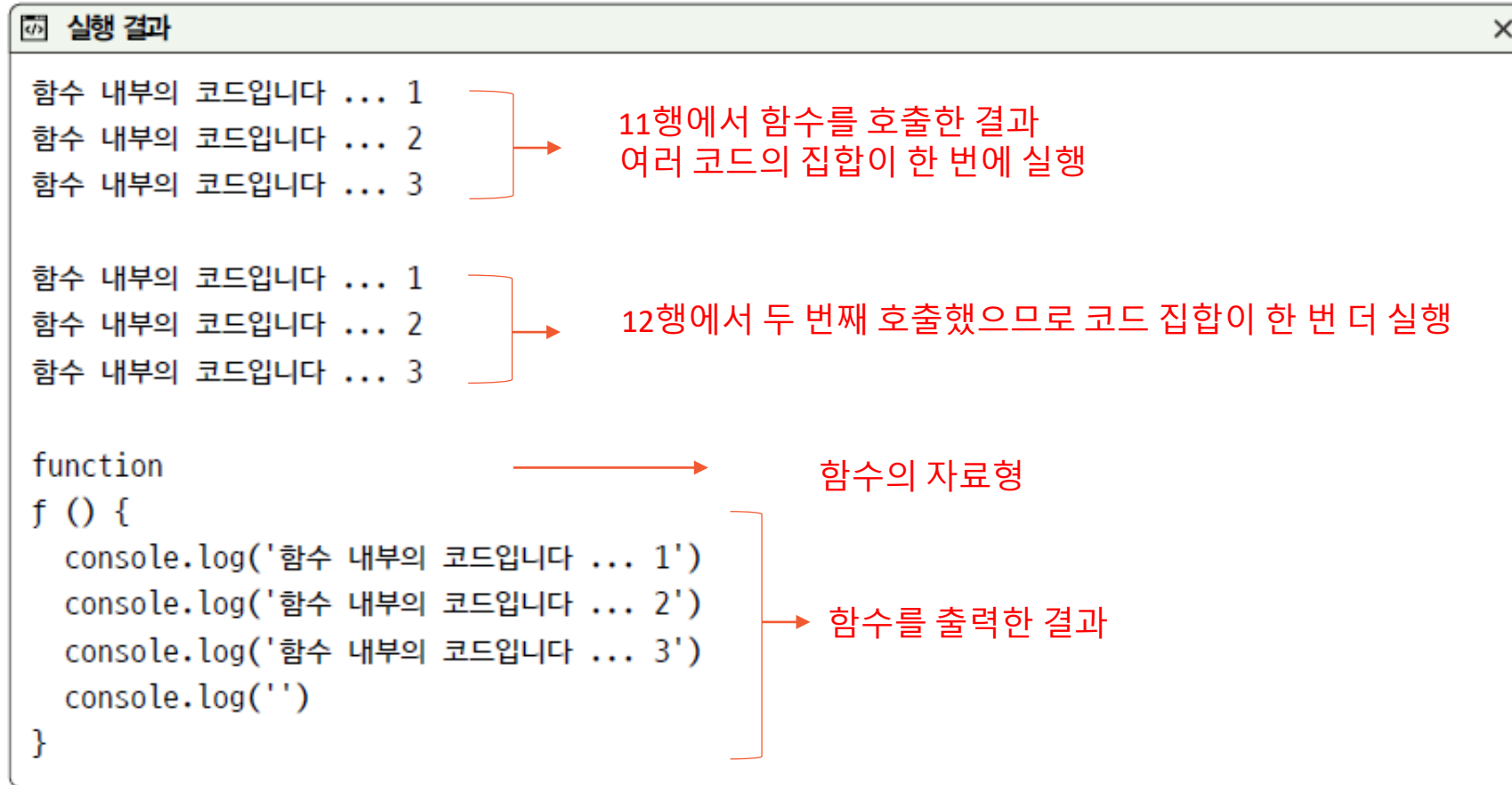
- 익명 함수 선언하기 (소스 코드 5-1-1.html)

```
01 <script>
02 // 변수를 생성합니다.
03 const 함수 = function () {
04   console.log('함수 내부의 코드입니다 ... 1')
05   console.log('함수 내부의 코드입니다 ... 2')
06   console.log('함수 내부의 코드입니다 ... 3')
07   console.log("")
08 }
09
10 // 함수를 호출합니다.
11 함수()           → 우리가 만든 함수도 기존의 alert(), prompt() 함수처럼 호출할 수 있음
12 함수()
13
14 // 출력합니다.
15 console.log(typeof 함수) → 함수의 자료형을 확인
16 console.log(함수)       → 함수 자체도 단순한 자료이므로 출력 가능
17 </script>
```

SECTION 5-1 함수의 기본 형태(2)

◦ 익명 함수

- 익명 함수 선언하기 (소스 코드 5-1-1.html) - 실행 결과



```
실행 결과
```

```
함수 내부의 코드입니다 ... 1
함수 내부의 코드입니다 ... 2
함수 내부의 코드입니다 ... 3
```

11행에서 함수를 호출한 결과
여러 코드의 집합이 한 번에 실행

```
함수 내부의 코드입니다 ... 1
함수 내부의 코드입니다 ... 2
함수 내부의 코드입니다 ... 3
```

12행에서 두 번째 호출했으므로 코드 집합이 한 번 더 실행

```
function
f () {
  console.log('함수 내부의 코드입니다 ... 1')
  console.log('함수 내부의 코드입니다 ... 2')
  console.log('함수 내부의 코드입니다 ... 3')
  console.log('')
}
```

함수의 자료형

함수를 출력한 결과

SECTION 5-1 함수의 기본 형태(3)

- 선언적 함수

- 선언적 함수는 이름을 붙여 생성

```
function 함수() {  
}
```

- 선언적 함수는 다음 코드와 같은 기능을 수행 (SECTION 5-2 좀 더 알아보기 참조)

```
let 함수 = function () { };
```

SECTION 5-1 함수의 기본 형태(4)

- 선언적 함수

- 선언적 함수 선언하기 (소스 코드 5-1-2.html)

```
01 <script>
02  // 함수를 생성합니다.
03  function 함수 () {
04    console.log('함수 내부의 코드입니다 ... 1')
05    console.log('함수 내부의 코드입니다 ... 2')
06    console.log('함수 내부의 코드입니다 ... 3')
07    console.log('')
08  }
09
10  // 함수를 호출합니다.
11  함수()
12  함수()
13
14  // 출력합니다.
15  console.log(typeof 함수)
16  console.log(함수)
17 </script>
```

SECTION 5-1 함수의 기본 형태(5)

- 선언적 함수

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

실행 결과

```
함수 내부의 코드입니다 ... 1
함수 내부의 코드입니다 ... 2
함수 내부의 코드입니다 ... 3

함수 내부의 코드입니다 ... 1
함수 내부의 코드입니다 ... 2
함수 내부의 코드입니다 ... 3

function
f 함수 () {
  console.log('함수 내부의 코드입니다 ... 1')
  console.log('함수 내부의 코드입니다 ... 2')
  console.log('함수 내부의 코드입니다 ... 3')
  console.log('')
}
```

이전과 다르게 함수에 이름이 붙어 있음

SECTION 5-1 함수의 기본 형태(6)

- 매개변수와 리턴값

- prompt() 함수의 매개변수와 리턴값

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

```
Prompt()
```

- 사용자 정의 함수의 매개변수와 리턴값

```
function 함수():void
```

```
함수()
```

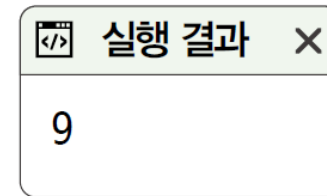
- 매개변수와 리턴값을 갖는 함수

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

SECTION 5-1 함수의 기본 형태(7)

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

```
01 <script>
02  // 함수를 선언합니다.
03  function f(x) {
04    return x * x
05  }
06
07  // 함수를 호출합니다.
08  console.log(f(3))
09 </script>
```





SECTION 5-1 함수의 기본 형태(8)

◦ 기본적인 함수 예제

- 윤년을 확인하는 함수 만들기
 - 조건 1) 4로 나누어 떨어지는 해는 윤년
 - 조건 2) 100으로 나누어 떨어지는 해는 윤년이 아님
 - 조건 3) 400으로 나누어 떨어지는 해는 윤년
- 윤년인지 확인하는 함수 (소스 코드 5-1-4.html)

```
01 <script>
02  function isLeapYear(year) {
03    return (year % 4 === 0) && (year % 100 !== 0) || (year % 400 === 0)
04  }
05
06  console.log(`2020년은 윤년일까? === ${isLeapYear(2020)}`)
07  console.log(`2010년은 윤년일까? === ${isLeapYear(2010)}`)
08  console.log(`2000년은 윤년일까? === ${isLeapYear(2000)}`)
09  console.log(`1900년은 윤년일까? === ${isLeapYear(1900)}`)
10 </script>
```


 실행 결과 

```
2020년은 윤년일까? === true
2010년은 윤년일까? === false
2000년은 윤년일까? === true
1900년은 윤년일까? === false
```

SECTION 5-1 함수의 기본 형태(9)

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

```
01 <script>
02  function sumAll(a, b) {
03    let output = 0
04    for (let i = a; i <= b; i++) {
05      output += i
06    }
07    return output
08  }
09
10  console.log(`1부터 100까지의 합: ${sumAll(1, 100)}`)
11  console.log(`1부터 500까지의 합: ${sumAll(1, 500)}`)
12 </script>
```

 실행 결과 ×


1부터 100까지의 합: 5050
1부터 500까지의 합: 125250

SECTION 5-1 함수의 기본 형태(10)

◦ 기본적인 함수 예제

- 최솟값을 구하는 함수 (소스 코드 5-1-6.html)

```
01 <script>
02 function min(array) {
03   let output = array[0]
04   for (const item of array) {
05     // 현재 output보다 더 작은 item이 있다면
06     if (output > item) {
07       // output의 값을 item으로 변경
08       output = item
09     }
10   }
11   return output
12 }
13
14 const testArray = [52, 273, 32, 103, 275, 24, 57]
15 console.log(`${testArray} 중에서`)
16 console.log(`최솟값 = ${min(testArray)}`)
17 </script>
```

 실행 결과 ×

52,273,32,103,275,24,57 중에서
최솟값 = 24

SECTION 5-1 함수의 기본 형태(11)

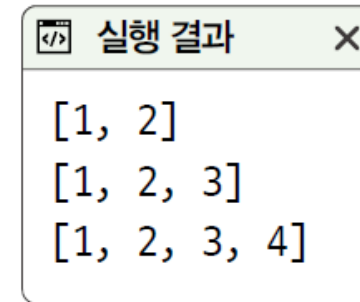
◦ 나머지 매개변수

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

function 함수 이름(...나머지 매개변수) {}

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

```
01 <script>
02  function sample(...items) {
03    console.log(items)
04  }
05
06  sample(1, 2)
07  sample(1, 2, 3)
08  sample(1, 2, 3, 4)
09 </script>
```

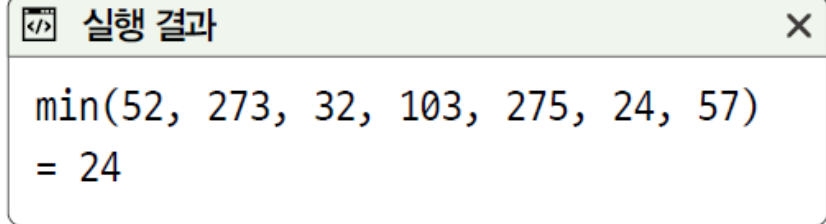


SECTION 5-1 함수의 기본 형태(12)

- 나머지 매개변수

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

```
01 <script>
02 // 나머지 매개변수를 사용한 함수 만들기
03 function min(...items) {
04   // 매개변수 items는 배열처럼 사용합니다.
05   let output = items[0]
06   for (const item of items) {
07     if (output > item) {
08       output = item
09     }
10   }
11   return output
12 }
13
14 // 함수 호출하기
15 console.log('min(52, 273, 32, 103, 275, 24, 57)')
16 console.log(`= ${min(52, 273, 32, 103, 275, 24, 57)}`)
17 </script>
```



실행 결과

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

SECTION 5-1 함수의 기본 형태(13)



◦ 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)

```
01 <script>
02  function sample(a, b, ...c) {
03    console.log(a, b, c)
04  }
05
06  sample(1, 2)
07  sample(1, 2, 3)
08  sample(1, 2, 3, 4)
09 </script>
```

 실행 결과 

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

SECTION 5-1 함수의 기본 형태(14)

◦ 나머지 매개변수

- Array.isArray() 메소드
 - 매개변수로 들어온 자료형이 배열인지 숫자인지 확인
- 매개변수의 자료형에 따라 다르게 작동하는 min() 함수 (소스 코드 5-1-10.html)

```
01 <script>
02 function min(first, ...rests) {
03   // 변수 선언하기
04   let output
05   let items
06
07   // 매개변수의 자료형에 따라 조건 분기하기
08   if (Array.isArray(first)) {
09     output = first[0]
10     items = first
11   } else if (typeof(first) === 'number') {
12     output = first
13     items = rests
14   }
15
16   // 이전 절에서 살펴보았던 최솟값 구하는 공식
17   for (const item of items) {
18     if (output > item) {
19       output = item
20     }
21   }
22   return output
23 }
24
25 console.log(`min(배열): ${min([52, 273, 32, 103, 275, 24, 57])}`)
26 console.log(`min(숫자, ...): ${min(52, 273, 32, 103, 275, 24, 57)}`)
27 </script>
```

실행 결과

```
min(배열): 24
min(숫자, ...): 24
```


SECTION 5-1 함수의 기본 형태(15)

◦ 나머지 매개변수

- Array.isArray() 메소드
 - 매개변수로 들어온 자료형이 배열인지 숫자인지 확인
- 매개변수의 자료형에 따라 다르게 작동하는 min() 함수 (소스 코드 5-1-10.html)

```
01 <script>
02 function min(first, ...rests) {
03     // 변수 선언하기
04     let output
05     let items
06
07     // 매개변수의 자료형에 따라 조건 분기하기
08     if (Array.isArray(first)) {
09         output = first[0]
10         items = first
```

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

중간 과정 생략

```
24
25 console.log(`min(배열): ${min([52, 273, 32, 103, 275, 24, 57])}`)
26 console.log(`min(숫자, ...): ${min(52, 273, 32, 103, 275, 24, 57)}`)
27 </script>
```

실행 결과


```
min(배열): 24
min(숫자, ...): 24
```

SECTION 5-1 함수의 기본 형태(16)

◦ 나머지 매개변수

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

```
01 <script>
02 // 단순히 매개변수를 모두 출력하는 함수
03 function sample(...items) {
04   console.log(items)
05 }
06
07 // 전개 연산자 사용 여부 비교하기
08 const array = [1, 2, 3, 4]
09
10 console.log('# 전개 연산자를 사용하지 않은 경우')
11 sample(array)
12 console.log('# 전개 연산자를 사용한 경우')
13 sample(...array)
14 </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(17)

◦ 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)

```
01 <script>
02 function earnings (name, wage=8590, hours=40) {
03   console.log(`# ${name} 님의 급여 정보`)
04   console.log(`- 시급: ${wage}원`)
05   console.log(`- 근무 시간: ${hours}시간`)
06   console.log(`- 급여: ${wage * hours}원`)
07   console.log("")
08 }
09
10 // 최저 임금으로 최대한 일하는 경우
11 earnings('구름')
12
13 // 시급 1만원으로 최대한 일하는 경우
14 earnings('별', 10000)
15
16 // 시급 1만원으로 52시간 일한 경우
17 earnings('인성', 10000, 52)
18 </script>
```

 실행 결과 ✕

```
# 구름 님의 급여 정보
- 시급: 8590원
- 근무 시간: 40시간
- 급여: 343600원

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

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

SECTION 5-1 함수의 기본 형태(19)

◦ 기본 매개변수

- 기본 매개변수를 추가한 윤년 함수 (소스 코드 5-1-13.html)

```
01 <script>
02 function isLeapYear(year=new Date().getFullYear()) {
03   console.log(`매개변수 year: ${year}`)
04   return (year % 4 === 0) && (year % 100 !== 0) || (year % 400 === 0)
05 }
06
07 console.log(`올해는 윤년일까? === ${isLeapYear()}`)
08 </script>
```

기본값을 이렇게 넣을 수도 있음

실행 결과

```
매개변수 year: 2020
올해는 윤년일까? === true
```

[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)

```
01 <script>
02 function sample() {
03   console.log(arguments)
04   for (let i = 0; i < arguments.length; i++) {
05     console.log(`${i}번째 요소: ${arguments[i]}`)
06   }
07 }
08
09 sample(1, 2)
10 sample(1, 2, 3)
11 sample(1, 2, 3, 4)
12 </script>
```

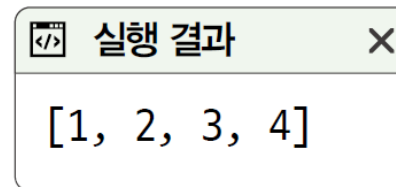
실행 결과

```
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 전개 연산자 구현하기

- 전개 연산자는 최신 버전의 자바스크립트에 추가된 기능
구 버전의 자바스크립트에서는 다음과 같이 apply() 함수를 사용한 굉장히 특이한 패턴의 코드를 사용
- 전개 연산자가 없던 구 버전에서 apply() 함수 사용하기 (소스 코드 5-1-15.html)

```
01 <script>
02 // 단순히 매개변수를 모두 출력하는 함수
03 function sample(...items) {
04   console.log(items)
05 }
06
07 // 전개 연산자 사용 여부 비교하기
08 const array = [1, 2, 3, 4]
09 console.log(sample.apply(null, array))
10 </script>
```



[좀 더 알아보기③] 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  
}
```

[마무리①]

◦ 7가지 키워드로 정리하는 핵심 포인트

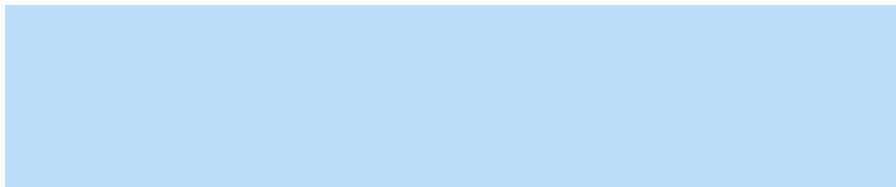
- 익명 함수란 이름이 없는 함수로 `function () { }` 형태로 만들
- 선언적 함수란 이름이 있는 함수로 `function 함수 이름 () { }` 형태로 만들
- 함수의 괄호 안에 넣는 변수를 매개변수라고 합니다. 매개변수를 통해 함수는 외부의 정보를 입력 받을 수 있음
- 함수의 최종적인 결과를 리턴값이라고 합니다. 함수 내부에 `return` 키워드를 입력하고 뒤에 값을 넣어서 생성
- 가변 매개변수 함수란 매개변수의 개수가 고정되어 있지 않은 함수를 의미. 나머지 매개변수(...)를 활용해서 만들
- 전개 연산자란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

[마무리②]

- 확인 문제

1. A부터 B까지 범위를 지정했을 때 범위 안의 숫자를 모두 곱하는 함수를 만들어보기

```
<script>
```



```
console.log(multiplyAll(1, 2))
```

```
console.log(multiplyAll(1, 3))
```

```
</script>
```



실행 결과



2

6

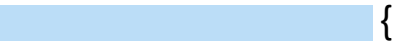
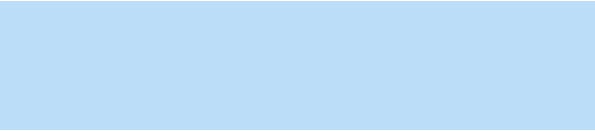
[마무리②]

◦ 확인 문제

2. 다음 과정에 따라 최대값을 찾는 max() 함수를 만들어보기

- 매개변수로 max([1, 2, 3, 4])와 같은 배열을 받는 max() 함수 만들기

①

```
<script>  
const max =  {  
  let output = array[0]  
  
    
  
  return output  
}  
console.log(max([1, 2, 3, 4]))  
</script>
```



[마무리③]

◦ 확인 문제

2. 다음 과정에 따라 최대값을 찾는 max() 함수를 만들어보기

- 매개변수로 max(1, 2, 3, 4)와 같이 숫자를 배열을 받는 max() 함수 만들기

②

```
<script>  
const max =  {  
  let output = array[0]  
  
    
  
  return output  
}  
console.log(max(1, 2, 3, 4))  
</script>
```


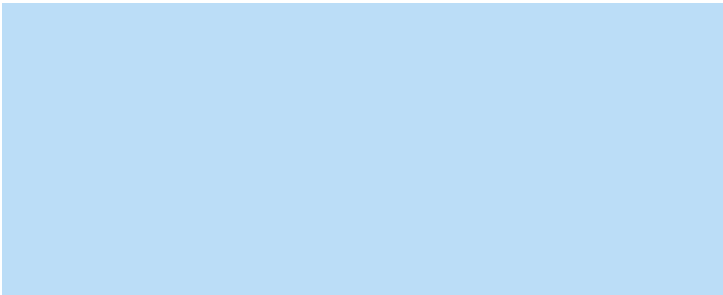
[마무리④]

◦ 확인 문제

2. 다음 과정에 따라 최대값을 찾는 max() 함수를 만들어보기

- max([1, 2, 3, 4]) 형태와 max(1, 2, 3, 4) 형태를 모두 입력할 수 있는 max() 함수 만들기

③

```
<script>
const max =  {
  let output
  let items
  
  return output
}
console.log(`max(배열): ${max([1,2,3,4])}`)
console.log(`max(숫자, ...): ${max(1,2,3,4)}`)
</script>
```

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 (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

- 화살표 함수는 function 키워드 대신 화살표(=>)를 사용하며, 다음과 같은 형태로 생성하는 간단한 함수

```
(매개변수) => {  
  불 표현식 || 불 표현식이 거짓일 때 실행할 문장
```

```
(매개변수) => 리턴값
```

- 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

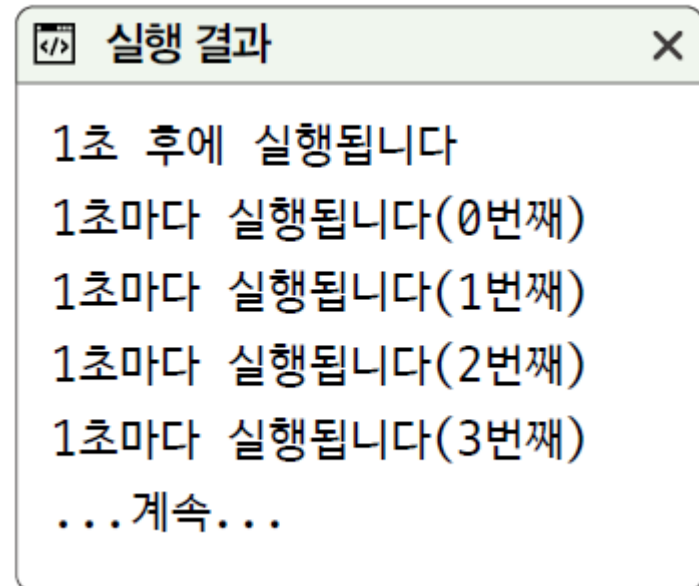
- 특정 시간마다 또는 특정 시간 이후에 콜백 함수를 호출할 수 있는 타이머(timer) 함수

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)

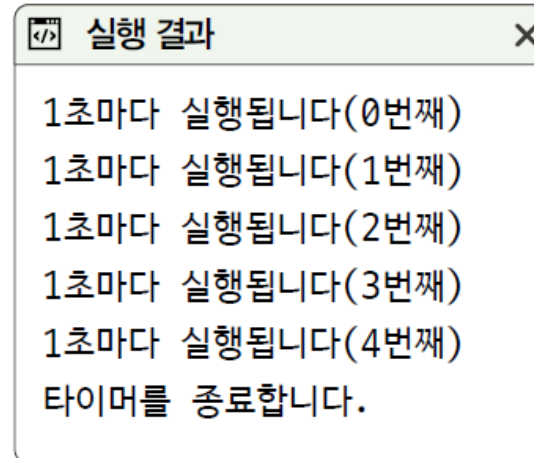
◦ 타이머 함수

- 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>
```



[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.`)
```

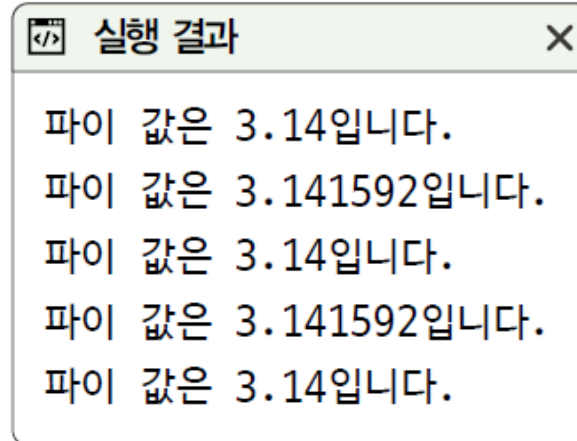
```
17 }
```

```
18 sample()
```

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

```
20 </script>
```

다른 블록에 속하므로 변수 이름 충돌
이 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 () {
```

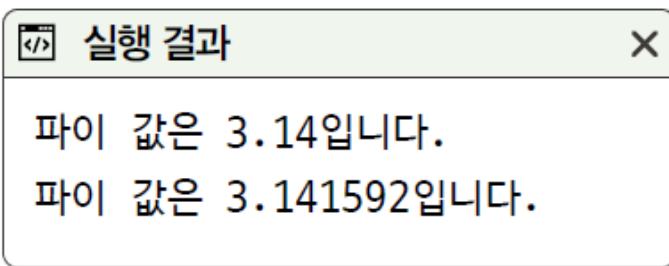
```
09   let pi = 3.141592
```

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

```
11 })()
```

```
12 </script>
```

즉시 호출 함수를 사용해
변수 이름 충돌 문제를 해결



[좀 더 알아보기③] 엄격 모드

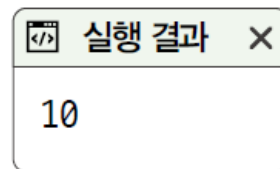
- 엄격 모드

- 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>
```

- 선언 없이 변수 사용 (소스 코드 5-2.12.html)

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

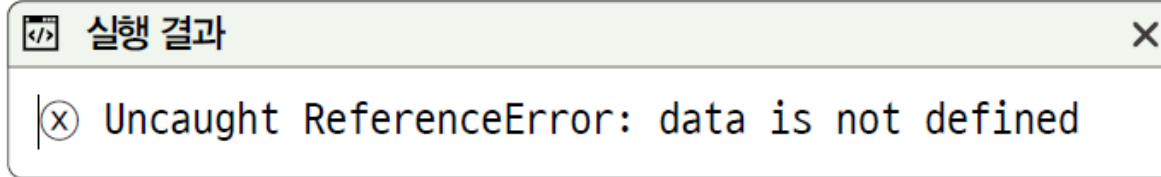


※ 엄격 모드에서는 이러한 코드를 사용할 수 없음
변수를 let 키워드 등으로 선언하지 않 았는데 사용했다고 곧바로 오류가 발생

[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

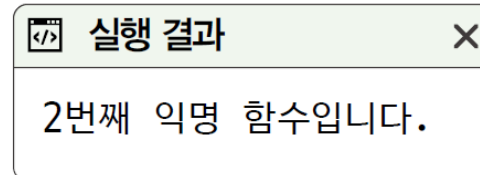
- 엄격 모드의 일반적인 사용 패턴

```
<script>
(function () {
  'use strict'
  문장
  문장
})();
</script>
```

[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)

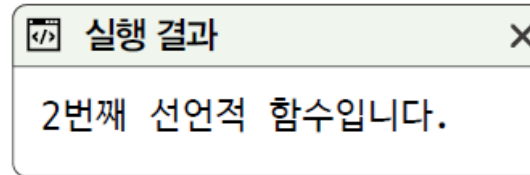
```
01 <script>
02  // 변수를 선언합니다.
03  let 익명함수
04
05  // 익명 함수를 2번 생성합니다.
06  익명함수 = function () {
07    console.log('1번째 익명 함수입니다.')
08  }
09  익명함수 = function () {
10    console.log('2번째 익명 함수입니다.')
11  }
12
13  // 익명 함수를 호출합니다.
14  익명함수()
15 </script>
```



[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)

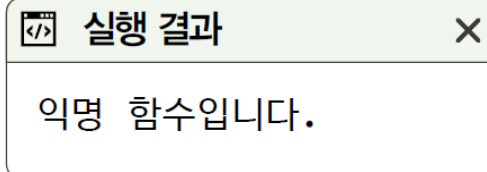
```
01 <script>
02 // 선언적 함수를 호출합니다.
03 선언적함수() → 선언적 함수를 생성하는 코드 앞에 입력
04
05 // 선언적 함수를 2번 생성합니다.
06 function 선언적함수 () {
07   console.log('1번째 선언적 함수입니다.')
08 }
09 function 선언적함수 () {
10   console.log('2번째 선언적 함수입니다.')
11 }
12 </script>
```



[좀 더 알아보기④] 익명 함수와 선언적 함수의 차이

- 선언적 함수와 익명 함수의 조합
 - 선언적 함수는 먼저 생성되고, 이후에 순차적인 코드 진행을 시작하면서 익명 함수를 생성. 따라서 다음과 같은 코드를 작성하면 코드의 순서와 관계 없이 "익명 함수입니다."라는 글자를 출력
 - 선언적 함수와 익명 함수의 조합 (소스 코드 5-2-16.html)

```
01 <script>
02 // 익명 함수를 생성합니다.
03 함수 = function () {
04   console.log('익명 함수입니다.')
05 }
06
07 // 선언적 함수를 생성하고 할당합니다.
08 function 함수 () {
09   console.log('선언적 함수입니다.')
10 }
11
12 // 함수를 호출합니다.
13 함수()
14 </script>
```





[좀 더 알아보기④] 익명 함수와 선언적 함수의 차이

- 블록이 다른 경우에 선언적 함수의 사용
 - 선언적 함수는 어떤 코드 블록(script 태그 또는 함수 등으로 구분되는 공간)을 읽어들일 때 먼저 생성
 - 블록이 다른 경우 선언적 함수의 사용 (소스 코드 5-2-17.html)

```
01 <script>
02  선언적함수()
03
04  function 선언적함수 () {
05    console.log('1번째 선언적 함수입니다.')
06  }
07 </script>
08 <script>
09  function 선언적함수 () {
10    console.log('2번째 선언적 함수입니다.')
11  }
12 </script>
13 <script>
14  선언적함수() → 블록 C
15 </script>
```

블록 A

블록 B

 실행 결과 

1번째 선언적 함수입니다.
2번째 선언적 함수입니다.

[좀 더 알아보기④] 익명 함수와 선언적 함수의 차이

- 과거 자바스크립트는 var이라는 키워드를 사용해서 변수를 선언
 - var 키워드는 이전 코드처럼 덮어쓰는 문제가 발생
 - 현대의 자바스크립트는 let 키워드와 const 키워드를 사용해서 변수와 상수를 선언
 - 이러한 키워드들은 위험을 원천적으로 차단하기 위해서 오류를 발생
- let 사용의 의미 (소스 코드 5-2-18.html)

```
01 <script>
02 // 익명 함수를 생성합니다.
03 let 함수 = function () {
04   console.log('익명 함수입니다.')
05 }
06
07 // 선언적 함수를 생성하고 할당합니다.
08 function 함수 () {
09   console.log('선언적 함수입니다.')
10 }
11
12 // 함수를 호출합니다.
13 함수()
14 </script>
```

실행 결과

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