

# 혼자 공부하는 자바스크립트



한국교통대학교 컴퓨터소프트웨어과  
최일준 교수  
cij0319@ut.ac.kr, cij0319@naver.com

# 이 책의 학습 목표

## ▪ CHAPTER 01: 자바스크립트 개요와 개발환경 설정

- 자바스크립트 개발환경 설치와 자바스크립트 프로그래밍 기본 용어 학습

## ▪ CHAPTER 02: 자료와 변수

- 프로그램 개발의 첫걸음. 자료형과 변수 학습

## ▪ CHAPTER 03: 조건문

- 프로그램의 흐름을 변화시키는 요소. 조건문의 종류를 알아보고 사용 방법을 이해

## ▪ CHAPTER 04: 반복문

- 배열의 개념과 문법을 익혀 while 반복문과 for 반복문 학습

## ▪ CHAPTER 05: 함수

- 다양한 형태의 함수를 만들기와 매개변수를 다루는 방법 이해

## ▪ CHAPTER 06: 객체

- 객체의 속성과 메소드, 생성, 관리하는 기본 문법 학습

## ▪ CHAPTER 07: 문서 객체 모델

- DOMContentLoaded 이벤트를 사용한 문서 객체 조작과 다양한 이벤트의 사용 방법 이해

## ▪ CHAPTER 08: 예외 처리

- 구문 오류와 예외를 구분하고, 예외 처리의 필요성과 예외를 강제로 발생시키는 방법을 이해

## ▪ CHAPTER 09: 클래스

- 객체 지향을 이해하고 클래스의 개념과 문법 학습

## ▪ CHAPTER 10: 리액트 라이브러리

- 리액트 라이브러리 사용 방법과 간단한 애플리케이션을 만드는 방법 학습

# 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

**함수 : 코드의 집합**

## 05-2 함수 고급

- 콜백 함수, 화살표 함수, 즉시 호출 함수, 엄격 모드
- 자바스크립트에서 함수는 자료이므로 변수에 할당할 수 있고, 함수를 함수의 매개변수로 전달해서 활용할 수 있음
- 함수를 매개변수로 전달하는 특성을 살펴봄

# 함수의 매개변수로 함수 전달하기

<> 5\_2\_1.html > ...

```
1 <script>
2 const 테스트 = function (a) {
3   console.log(a)
4 }
5
6 테스트(10)
7 테스트("안녕하세요")
8 테스트(true)
9 </script>
```

10

안녕하세요

true

content loaded

>

<> 5\_2\_2.html > ...

```
1 <script>
2   const 테스트 = function (a) {
3     console.log(a)
4   }
5
6   테스트(10)
7   테스트("안녕하세요")
8   테스트(true)
9
10  const 함수 = function () {
11    console.log("안녕하세요")
12  }
13  테스트(함수)
14 </script>
```

10

안녕하세요

true

```
f () {
  console.log("안녕하세요")
}
```

content loaded

>

# 함수의 매개변수로 함수 전달하기

```
<> 5_2_3.html > ...
1 <script>
2   const 테스트 = function (a) {
3     | a()
4   }
5
6   const 함수 = function () {
7     | console.log("안녕하세요")
8   }
9   테스트(함수)
10 </script>
```

안녕하세요

content loaded

```
<> 5_2_4.html > ...
1 <script>
2   const 테스트 = function (a) {
3     | a()
4   }
5
6   const 함수 = function () {
7     | console.log("안녕하세요")
8   }
9   테스트(함수)
10  테스트(10)
11  테스트("안녕하세요")
12 </script>
```

안녕하세요

✖ Uncaught TypeError: a is not a function  
at 테스트 (5\_2\_4.html:3:5)  
at 5\_2\_4.html:10:3

content loaded

```
<> 5_2_5.html > ...
1 <script>
2   const 테스트 = function (콜백함수) {
3     | 콜백함수()
4   }
5
6   const 함수 = function () {
7     | console.log("안녕하세요")
8   }
9
10  테스트(함수)
11 </script>
```

안녕하세요

content loaded

**a is not function**

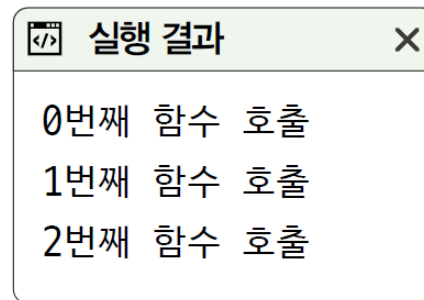
-> a가 함수가 아닌데 왜 호출하려고 하고 있나?

## SECTION 5-2 함수 고급(1)

### 콜백(callback) 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) -> 콜백함수에 매개변수 전달하기

```
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>
```





# 콜백(callback) 함수 사용 예

```
5_1_1.html > ...
1 <script>
2 const 테스트 = function (콜백함수){
3   콜백함수(10)
4 }
5
6 const 함수 = function (콜백함수의_매개변수){
7   console.log(`${콜백함수의_매개변수}번째 안녕하세요`)
8 }
9 테스트(함수)
10 </script>
```

10번째 안녕하세요

content loaded

```
5_1_2.html > ...
1 <script>
2   const 테스트 = function (콜백함수){
3     for (let i = 0; i < 5; i++){
4       콜백함수(i)
5     }
6   }
7
8   const 함수 = function (콜백함수의_매개변수){
9     console.log(`${콜백함수의_매개변수}번째 안녕하세요`)
10   }
11   테스트(함수)
12 </script>
```

0번째 안녕하세요

1번째 안녕하세요

2번째 안녕하세요

3번째 안녕하세요

4번째 안녕하세요

content loaded

## SECTION 5-2 함수 고급(2)

### 콜백 함수

- 콜백 함수(2): **익명 함수 사용하기** (소스 코드 5-2-2.html) -> 콜백함수에 매개변수 전달하기

```
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, array를 갖는 콜백 함수를 사용

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

# 콜백함수 호출- 배열 내부의 요소

```
<> 5_1_3.html > ...
1  <script>
2  |   const 테스트 = function (배열, 콜백함수){
3  |       for (const 값 of 배열){
4  |           콜백함수(값)
5  |       }
6  |   }
7
8  |   const 함수 = function (콜백함수의_매개변수){
9  |       console.log(`${콜백함수의_매개변수}번째 안녕하세요`)
10 |   }
11 |   테스트([52, 273, 103, 32], 함수)
12 </script>
```

52번째 안녕하세요

273번째 안녕하세요

103번째 안녕하세요

32번째 안녕하세요

content loaded

>

# 배열의 콜백함수를 사용하는 메소드

## forEach(), filter(), map() 메소드

<> 5\_1\_4.html > ...

```
1 <script>
2   const 배열 = [273, 52, 103, 32, 57]
3   배열.forEach(function (value, index, array){
4     | console.log(value, index, array)
5   })
6 </script>
```

273 0 ▶ (5) [273, 52, 103, 32, 57]

52 1 ▶ (5) [273, 52, 103, 32, 57]

103 2 ▶ (5) [273, 52, 103, 32, 57]

32 3 ▶ (5) [273, 52, 103, 32, 57]

57 4 ▶ (5) [273, 52, 103, 32, 57]

content loaded

<> 5\_1\_5.html > ...

```
1 <script>
2   const 배열 = [273, 52, 103, 32, 57]
3   배열.forEach(function (value, index){
4     | console.log(`${index}번째의 값은 ${value}`)
5   })
6 </script>
```

0번째의 값은 273

1번째의 값은 52

2번째의 값은 103

3번째의 값은 32

4번째의 값은 57

content loaded

>

## 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

# 콜백 함수를 활용하는 함수: map()

```
<> 5_2_6.html > ...  
1  <script>  
2    let 배열 = [273, 52, 103, 32, 57]  
3    배열 = 배열.map(function (value, index){  
4      |   return value + "!!"  
5    |   })  
6  
7    console.log(배열)  
8  </script>
```

```
▶ (5) ['273!!', '52!!', '103!!', '32!!', '57!!']
```

```
content loaded
```

```
>
```



## SECTION 5-2 함수 고급(6)

### 콜백 함수

- 콜백 함수를 활용하는 함수: `filter()`
- `filter()` 메소드는 콜백 함수에서 리턴하는 값이 `true`인 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

# 콜백 함수를 활용하는 함수: filter() 예제

```
<> 5_2_9.html > ...
1  <script>
2
3  const 배열 = [273, 52, 103, 32, 57]
4  console.log(배열.filter(function (value, index){
5  |   return true
6  |}))
7
8  console.log(배열.filter(function (value, index){
9  |   return false
10 |}))
11
12 </script>
```

▶ (5) [273, 52, 103, 32, 57]

▶ []

content loaded

>

```
<> 5_2_10.html > ...
1  <script>
2
3  let 배열 = [273, 52, 103, 32, 57]
4  배열 = 배열.filter(function (value, index){
5  |   return value % 2 === 0
6  |})
7
8  console.log(배열)
9
10 </script>
```

▶ (2) [52, 32]

content loaded

>

# 콜백함수 정리

<> 5\_2\_11.html > ...

```
1 <script>
2
3   const 배열 = [273, 52, 103, 32, 57]
4   배열.forEach(function (value, index){
5     console.log(`${index}번째의 값은 ${value}`)
6   })
7
8   let 배열 = [273, 52, 103, 32, 57]
9   배열 = 배열.filter(function (value, index){
10    return value % 2 === 0
11  })
12  console.log(배열)
13
14  let 배열 = [273, 52, 103, 32, 57]
15  배열 = 배열.map(function (value, index){
16    return value + "!!"
17  })
18  console.log(배열)
19
20 </script>
```

<> 5\_2\_12.html > ...

```
1 <script>
2   const myForEach = function (배열, 콜백함수) {
3     for (let i = 1; i < 배열.length; i++){
4       const element = 배열[i];
5       콜백함수(element, i, 배열)
6     }
7   }
8
9   const myFilter = function (배열, 콜백함수) {
10    const output = []
11    for (let i = 1; i < 배열.length; i++){
12      const element = 배열[i];
13      if (콜백함수(element, i, 배열)){
14        output.push(element)
15      }
16    }
17    return output
18  }
19 </script>
```

## 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

```
# 화살표 함수  
(매개변수) => { 본문 }
```

실행 결과

0
4
16
36
64

# 화살표 함수 예제

```
<> 5_2_6.html > ...
1  <script>
3    let array = [273, 52, 103, 32, 57]
4
5    array = array.filter(function (value, index){
6      |   return value % 2 === 0
7    })
8
9    array = array.filter((value, index) => value % 2 === 0)
10   console.log(array)
11
12   array = array.map(function (value, index){
13     |   return value + "!!"
14   })
15
16   array = array.map((value, index) => value + "!!")
17   console.log(array)
18
19  </script>
```

▶ (2) [52, 32]

▶ (2) ['52!!!!', '32!!!!']

content loaded



## SECTION 5-2 함수 고급(8)

### ◦ 타이머 함수

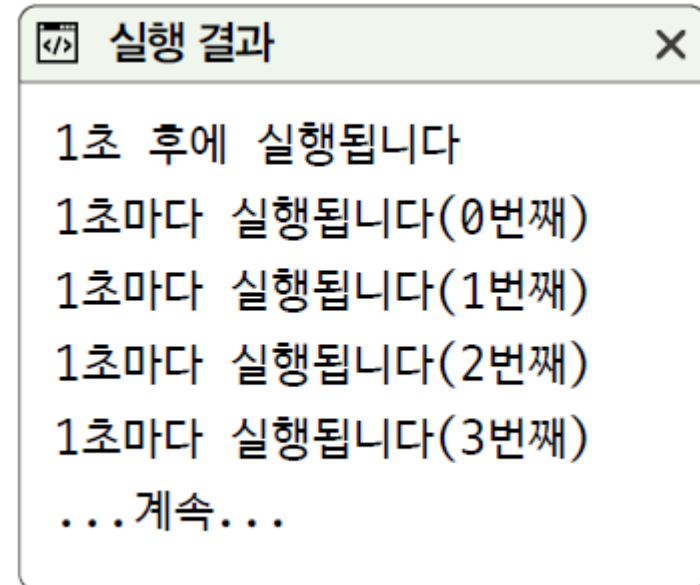
- 특정 시간마다 또는 특정 시간 이후에 콜백 함수를 호출할 수 있는 타이머(timer) 함수

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 →



# 타이머 함수 예제

```
<> 5_2_8.html > ...  
1 <script>  
2   const a = setTimeout(function () {  
3     | console.log('setTimeout 함수 입니다!.')  
4   }, 1000) //1초 = 1*1000  
5   const b = setInterval(function () {  
6     | console.log('setInterval 함수 입니다!.')  
7   }, 1000) //1초 = 1*1000  
8  
9 </script>
```

```
content loaded  
setTimeout 함수 입니다!.  
24 setInterval 함수 입니다!.  
>
```

시계를 만들 때, 움직임을 구현할 때 등에 사용

## SECTION 5-2 함수 고급(9)

### ◦ 타이머 함수

- 타이머를 종료하고 싶을 때는 `clearTimeout()` 함수와 `clearInterval()` 함수를 사용, = 타이머 제거 함수

함수 이름	설명
<code>clearTimeout(타이머_ID)</code>	<code>setTimeout()</code> 함수로 설정한 타이머를 제거
<code>clearInterval(타이머_ID)</code>	<code>setInterval()</code> 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>
```

`setTimeout`은 `clearTimeout` 으로  
`setInterval`은 `clearInterval`로 타이머를 끄

실행 결과

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



# 타이머 취소하기 다른 예제

<> 5\_2\_7.html > ...

```
1 <script>
2 const a = setTimeout(function () {
3   console.log('setTimeout 함수 입니다!..')
4 }, 1000) //1초 = 1*1000
5 const b = setInterval(function () {
6   console.log('setInterval 함수 입니다!..')
7 }, 1000) //1초 = 1*1000
8
9 console.log(a,b)
10
11 clearTimeout(a)
12 clearInterval(b)
13 </script>
```

1 2

content loaded

>

# [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>

```
1 (function () {  
2  
3  
4 })()  
5  
6 (() => {  
7  
8 })()
```

```
const a = function () {}  
a()
```

```
const a =  
(function () {} )()
```

함수를 만들고 즉시 호출하는 함수  
→ 즉시 호출 함수  
IIFE (Immediately Invoked Function Expression)라고 부르기도 합니다

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

실행 결과

파이 값은 3.14입니다.  
파이 값은 3.141592입니다.  
파이 값은 3.14입니다.  
파이 값은 3.141592입니다.  
파이 값은 3.14입니다.

```
< 5_1.html > ...
1  <script>
2  |  (function () {
3  |      const a = 10
4  |      console.log(a)
5  |  })()
6  </script>
7  <script>
8  |  (function () {
9  |      const a = 20
10 |      console.log(a)
11 |  })()
12 </script>
```

10

20

content loaded

## [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 () {
```

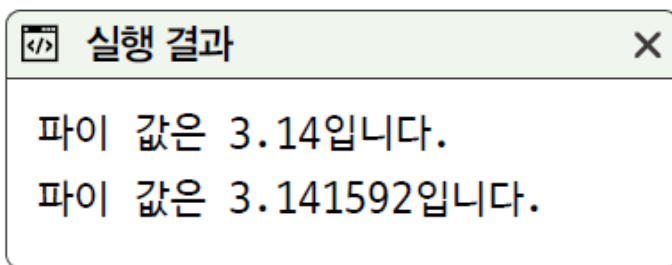
```
09   let pi = 3.141592
```

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

```
11 })()
```

```
12 </script>
```

즉시 호출 함수를 사용해  
변수 이름 충돌 문제를 해결



## [좀 더 알아보기③] 엄격 모드(strict mode)

- 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>
```

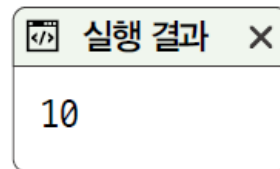
---

- 선언 없이 변수 사용 (소스 코드 5-2.12.html)

---

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

---

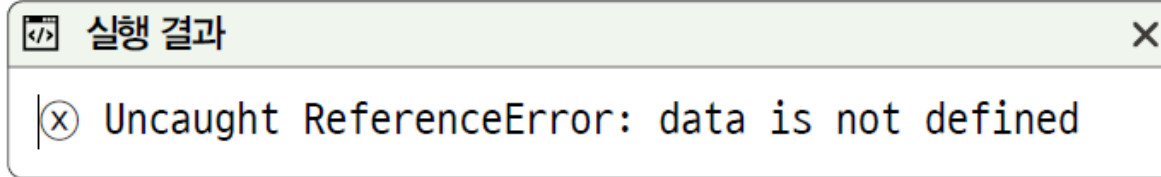


※ 엄격 모드에서는 이러한 코드를 사용할 수 없음  
변수를 let 키워드 등으로 선언하지 않았는데 사용했다고 곧바로 오류가 발생

## [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>
```

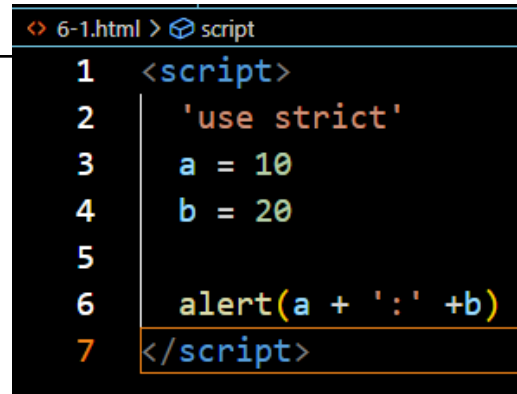


- 모질라 엄격 모드 문서 참조

URL [https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Strict\\_mode](https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Strict_mode)

- 엄격 모드의 일반적인 사용 패턴

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

A screenshot of a code editor window titled '6-1.html > script'. It shows a JavaScript code block with the following content:

```
1 <script>
2   'use strict'
3   a = 10
4   b = 20
5
6   alert(a + ':' + b)
7 </script>
```

A screenshot of a web browser's developer console. It shows a red 'X' icon followed by the text 'Uncaught ReferenceError: a is not defined at 6-1.html:3:5'.

content loaded

원래 개발은 무슨 짓을 하는게 눈에 보이는데,  
자바스크립트는 (function() {}), 'use strict'처럼  
"애는 뜬금없이 뭐냐?"하는 부분이 좀 많아서  
다른 언어 하시다가 넘어오신 분들이 많이 당황하는 언어입니다.

## [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>
```

자바스크립트는 네임스페이스(namespace)라는 문법이 따로 없어서  
이름 충돌이 쉽게 발생할 수 있는 언어입니다.  
그래서 이후에 활용하는 객체로 네임스페이스를 유사적으로 만들어 활용합니다.  
그래서 선언적 함수를 잘 안 씁니다.

실행 결과

2번째 익명 함수입니다.

# 1)익명 함수와 2)선언적 함수의 비교 예시

선언적 함수는 전체 코드를 읽기 전에 선언한 순서대로 만들어집니다.

```
<> 5_3.html > ...
1 <script>
2 let 함수 = function () {
3   console.log('A 함수입니다.')
4 }
5 함수 = function () {
6   console.log('B 함수입니다.')
7 }
8 함수 = function () {
9   console.log('C 함수입니다.')
10 }
11 함수()
12 </script>
```

C 함수입니다.

content loaded



```
<> 5_4.html > ...
1 <script>
2   function 함수 () {
3     console.log('A 함수입니다.')
4   }
5
6   function 함수 () {
7     console.log('B 함수입니다.')
8   }
9   function 함수 () {
10    | console.log('C 함수입니다.')
11   }
12   함수()
13 </script>
```

C 함수입니다.

content loaded

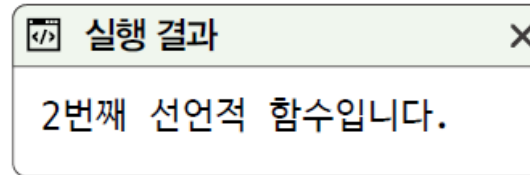




## [좀 더 알아보기④] 익명 함수와 선언적 함수의 차이

- 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>
```

실행 결과

익명 함수입니다.

```
> 함수()
함수 = function () {
  console.log('익명 함수입니다.')
}
function 함수() {
  console.log('선언적 함수입니다.')
}
함수()
선언적 함수입니다.
익명 함수입니다.
< undefined
```

② 애가 나중에 실행됩니다.

① 애가 먼저 만들어지고



## [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

# 익명 함수가 현대에서는 많이 사용됨.

```
<> 6-1.html > ...
1  <script>
2  | 함수()
3
4  | 함수 = function () {
5  | | console.log('익명 함수입니다.')
6  | }
7  </script>
8  <script>
9  | function 함수() {
10 | | console.log('선언적 함수입니다.')
11 | }
12
13 | 함수()
14 </script>
```

✖ Uncaught ReferenceError: 함수 is not defined  
at 6-1.html:2:3

선언적 함수입니다.

content loaded

```
<> 6-1.html > ...
1  <script>
2  | 함수()
3
4  | function 함수 () {
5  | | console.log('선언적 함수입니다.')
6  | }
7  </script>
8  <script>
9  | 함수()
10
11 | 함수 = function() {
12 | | console.log('익명 함수입니다.')
13 | }
14
15 </script>
```

② 선언적 함수입니다.

content loaded

>

## [마무리①]

- 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 모드는 실수를 줄일 수 있는 방법  
'use strict'라는 문자열을 블록 가장 위쪽에 배치해서 사용

## [마무리②]

### ◦ 확인 문제

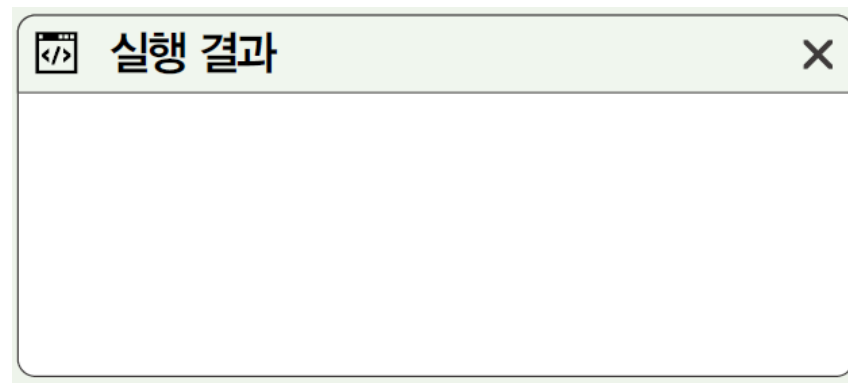
1. filter 함수의 콜백 함수 부분을 채워서 ① 홀수만 추출, ② 100 이하의 수만 추출, ③ 5로 나눈 나머지가 0인 수만 추출하고, 코드의 실행 결과를 적어 보기

---

```
<script>  
  // 변수를 선언합니다.  
  let numbers = [273, 25, 75, 52, 103, 32, 57, 24, 76]  
  // 처리합니다.
```

```
  // 출력합니다.  
  console.log(numbers)  
</script>
```

---



# 1번 문제 정답

```
<> 6_3.html > ...
1  <script>
2    // 변수를 선언합니다.
3    let numbers = [273, 25, 75, 52, 103, 32, 57, 24, 76]
4
5    // 홀수만 추출
6    numbers = numbers.filter((value) => value % 2 === 1)
7    console.log('# 홀수만 추출')
8    console.log(numbers)
9
10   // 100 이하의 수만 추출
11   numbers = numbers.filter((value) => value <= 100)
12   console.log('# 100 이하의 수만 추출')
13   console.log(numbers)
14
15   // 5로 나눈 나머지가 0인 수만 추출
16   numbers = numbers.filter((value) => value % 5 === 0)
17   console.log('# 5로 나눈 나머지가 0인 수만 추출')
18   console.log(numbers)
19 </script>
```

# 홀수만 추출

▶ (5) [273, 25, 75, 103, 57]

# 100 이하의 수만 추출

▶ (3) [25, 75, 57]

# 5로 나눈 나머지가 0인 수만 추출

▶ (2) [25, 75]

content loaded

>

```
<> 6_3_1.html > script
1  <script>
2    let numbers = [273, 25, 75, 52, 103, 32, 57, 24, 76]
3
4    numbers = numbers.filter((value) => value % 2 === 1)
5    console.log(numbers)
6    numbers = numbers.filter((value) => value <= 100)
7    console.log(numbers)
8    numbers = numbers.filter((value) => value % 5 === 0)
9    console.log(numbers)
10
11 </script>
```

▶ (5) [273, 25, 75, 103, 57]

▶ (3) [25, 75, 57]

▶ (2) [25, 75]

content loaded

>



## [마무리③]

### ◦ 확인 문제

2. 이전에 반복문 부분에서 살펴보았던 다음과 같은 코드를 **배열의 forEach 메소드를 사용하는 형태로 변경하기** 오른쪽의 실행 결과가 나오도록 해야 함

```
<script>
const array = ['사과', '배', '귤', '바나나']
console.log('# for in 반복문') -> 인덱스 추출
for (const i in array) {
  console.log(i)
}
console.log('# for of 반복문') -> 값을 추출
for (const i of array) {
  console.log(i)
}
</script>
```

for 반복문에서 사용할 수 있던 break를  
forEach() 메서드에서는 구현이 안 되므로  
break를 해야하는 경우에는 for 반복문을 씁니다.  
[근데 그래야 하는 상황의 비율이 생각보다 많지는 않습니다].

```
실행 결과
# for in 반복문
0
1
2
3
# for of 반복문
사과
배
귤
바나나
```

## 2번 문제 정답

<> 6\_2.html > ...

```
1 <script>
2 const array = ['사과', '배', '귤', '바나나']
3 console.log('# for in 반복문')
4
5 array.forEach(function (value, index){
6 |   console.log(index)
7 | })
8
9 console.log('# for of 반복문')
10
11 array.forEach((value, index) => {
12 |   console.log(value)
13 | })
14 </script>
15
```

# for in 반복문

0

1

2

3

# for of 반복문

사과

배

귤

바나나

content loaded

오늘도 고생하셨습니다.