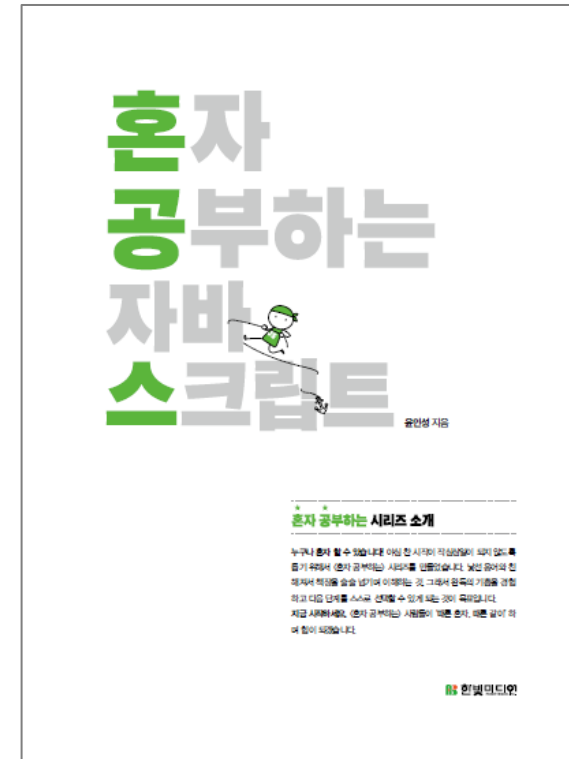


혼자 공부하는 자바스크립트



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 만들기과 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 04: 반복문

SECTION 4-1 배열

SECTION 4-2 반복문



CHAPTER 04 반복문

배열의 개념과 문법을 익혀 while 반복문과 for 반복문 학습

SECTION 4-1 배열(1)

- 배열(array): 여러 자료를 묶어서 활용할 수 있는 특수한 자료

```
> const str = '안녕하세요'
```

```
> str[2]
```

①

```
> str[str.length - 1]
```

②

str

안	녕	하	세	요
[0]	[1]	[2]	[3]	[4]

↑
str[2]

↑
str[str.length-1]
↓
5

SECTION 4-1 배열(2)

배열 만들기

[요소, 요소, 요소, ... ,요소]

```
> const array = [273, 'String', true, function () { }, {}, [273, 103]]
```

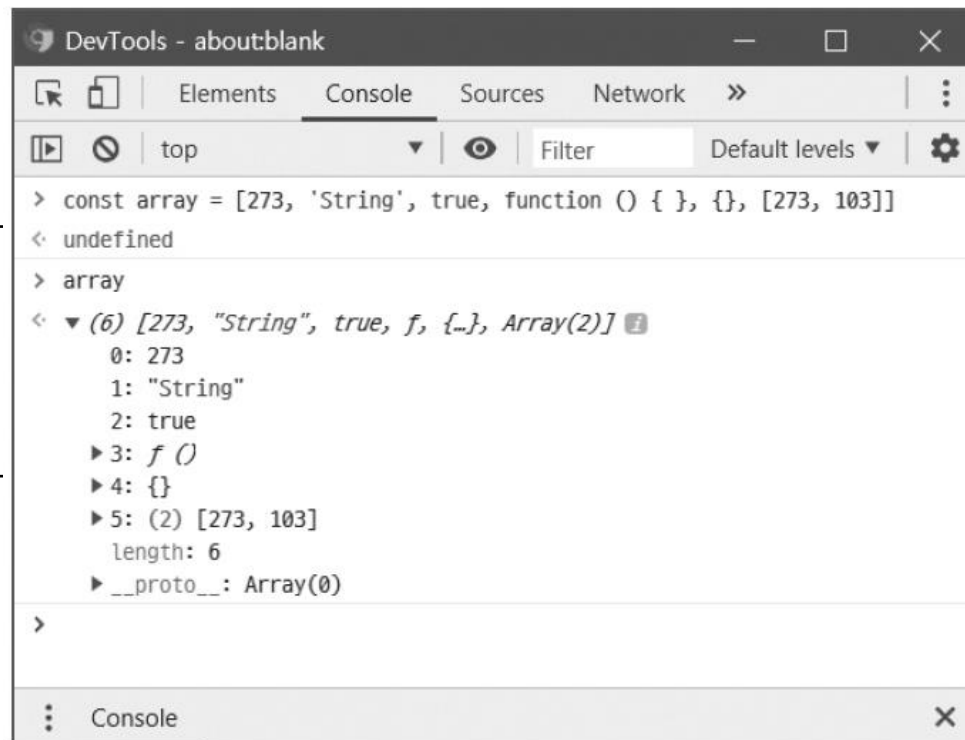
undefined

```
> array Enter
```

```
(6) [273, "String", true, f, {...}, Array(2)]
```

요소 개수

요소



구글 크롬 개발자 도구의 Console에서 코드를 실행할 때 출력된 배열 결과 왼쪽에 드롭 다운 버튼 ▶ 클릭하면 0번째에 273, 1번째에 "String", 2번째에 true 등의 값을 확인 할 수 있음

SECTION 4-1 배열(3)

- 배열 요소에 접근하기

배열[인덱스]

```
> const numbers = [273, 52, 103, 32]
```

```
undefined
```

```
> numbers[0]
```

```
273
```

```
> numbers[1]
```

```
52
```

```
> numbers[1 + 1]
```

```
103
```

```
> numbers[1 * 3]
```

```
32
```

① 배열은 여러 개의 요소를 갖기 때문에 일반적으로 배열 이름을 복수형으로(number → numbers)

② numbers[1 + 1], numbers[1 * 3]처럼 대괄호 안에 계산식을 넣을 수도 있음

SECTION 4-1 배열(4)

- 배열 요소 개수 확인하기

배열.length

> const fruits = ['배', '사과', '키위', '바나나']
undefined

→ 배열 이름을 복수형으로 작성

> fruits.length
4

→ 4 배열 fruits에 4개의 요소가 들어있으므로 4를 출력

> fruits[fruits.length - 1]
"바나나"

→ fruits[4-1], 배열의 3번째 요소인 "바나나"를 출력

SECTION 4-1 배열(5)

배열 뒷부분에 요소 추가하기

- push() 메소드를 사용해 배열 뒷부분에 요소 추가하기

배열.push(요소)

```
> const todos = ['우유 구매', '업무 메일 확인하기', '필라테스 수업']
```

```
undefined
```

```
> todos
```

```
(3) ["우유 구매", "업무 메일 확인하기", "필라테스 수업"]
```

```
> todos.push('저녁 식사 준비하기')
```

```
4
```

```
> todos.push('피아노 연습하기')
```

```
5
```

```
> todos
```

```
(5) ["우유 구매", "업무 메일 확인하기", "필라테스 수업", "저녁 식사 준비하기", "피아노 연습하기"]
```

push() 메소드로 요소가 추가되어 기존 요소
개수에서 추가된 요소 개수가 출력

↓
뒷부분에 2개의 요소가 추가

SECTION 4-1 배열(6)

배열 뒷부분에 요소 추가하기

- 인덱스를 사용해 배열 뒷부분에 요소 추가하기

```
> const fruitsA = ['사과', '배', '바나나']
```

```
Undefined
```

```
> fruitsA[10] = '귤'
```

```
"귤"
```

```
> fruitsA
```

```
(11) ["사과", "배", "바나나", empty × 7, "귤"]
```

```
> const fruitsB = ['사과', '배', '바나나']
```

```
Undefined
```

```
> fruitsB[fruitsB.length] = '귤'.
```

```
"귤"
```

```
> fruitsB
```

```
(4) ["사과", "배", "바나나", "귤"]
```

—————→ 배열 10번째 인덱스에 "귤"을 추가

—————→ fruitsB의 요소는 3개이므로 fruitsB[3]에 "귤"을 추가

SECTION 4-1 배열(7)

- 배열 요소 제거하기

- 인덱스로 요소 제거하기

```
배열.splice(인덱스, 제거할 요소의 개수)
```

```
> const itemsA = ['사과', '배', '바나나']  
undefined
```

```
> itemsA.splice(2, 1).  
["바나나"]
```



배열의 2번째 인덱스로부터 1개 요소를 제거
제거된 요소가 배열로 리턴

```
> itemsA  
(2) ["사과", "배"]
```



배열의 값을 확인해보면 요소가 제거된 것을 알 수 있음

SECTION 4-1 배열(8)

배열 요소 제거하기

값으로 요소 제거하기

```
const 인덱스 = 배열.indexOf(요소)  
배열.splice(인덱스, 1)
```

```
> const itemsB = ['사과', '배', '바나나']
```

```
undefined
```

```
> const index = itemsB.indexOf('바나나')
```

```
Undefined
```

```
> index
```

```
2
```

→ 배열 내부에 바나나가 있으므로 해당 요소의 인덱스를 출력

```
> itemsB.splice(index, 1)
```

```
["바나나"]
```

→ 배열의 2번째 인덱스에 있는 1개의 요소를 제거

```
> itemsB
```

```
(2) ["사과", "배"]
```

→ 배열에서 바나나가 제거

```
> itemsB.indexOf('바나나')
```

```
-1
```

→ 바나나는 배열에 없으므로 -1을 출력

SECTION 4-1 배열(8)

- 배열의 특정 위치에 요소 추가하기

- 값으로 요소 제거하기

```
배열.splice(인덱스, 0, 요소)
```

```
> const itemsD = ["사과", "귤", "바나나", "오렌지"]  
Undefined
```

```
> itemsD.splice(1, 0, "양파")  
[]
```

```
> itemsD  
(5) ["사과", "양파", "귤", "바나나", "오렌지"]
```

→ 1번째 인덱스에 양파가 추가

[좀 더 알아보기①]

- 자료 처리 연산자, 함수, 메소드는 크게 비파괴적 처리와 파괴적 처리로 구분

- 비파괴적 처리

```
> const a = '안녕'
> const b = '하세요'
> const c = a + b
> c
"안녕하세요"
> a
"안녕"
> b
"하세요"
```

변수를 선언

문자열을 연결하는 처리 수행

원본 내용이 변경되지 않음

- 메모리가 여유로운 현대의 프로그래밍 언어와 라이브러리는 자료 보호를 위해서 대부분 비파괴적 처리를 수행

[좀 더 알아보기②]

- 자료 처리 연산자, 함수, 메소드는 크게 비파괴적 처리와 파괴적 처리로 구분
 - 파괴적 처리

```
> const array = ["사과", "배", "바나나"]  
> array.push("귤")  
4  
> array  
(4) ["사과", "배", "바나나", "귤"]
```

변수를 선언
배열 뒷부분에 요소를 추가하는 처리
원본(array) 내용이 변경됨

[마무리①]

◦ 4가지 키워드로 정리하는 핵심 포인트

- 여러 개의 변수를 한 번에 선언해 다룰 수 있는 자료형을 배열이라 함
- 배열 내부에 있는 값은 요소
- 비파괴적 처리란 처리 후에 원본 내용이 변경되지 않는 처리를 의미
- 파괴적 처리란 처리 후에 원본 내용이 변경되는 처리를 의미

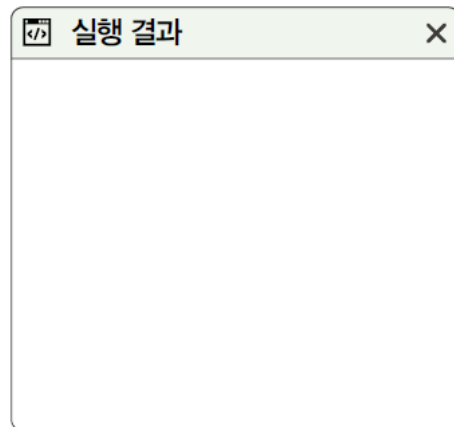
◦ 확인 문제

1. 다음 배열들의 2번째 인덱스에 있는 값을 찾아보기

① ["1", "2", "3", "4"] ② ["사과", "배", "바나나", "귤", "감"] ③ [52, 273, 32, 103, 57]

2. 다음 코드의 실행 결과를 예측

```
<script>  
  const array = [1, 2, 3, 4]  
  console.log(array.length)  
  console.log(array.push(5))  
</script>
```



[마무리②]

◦ 확인 문제

3. 다음 표시된 함수들이 파괴적 처리를 하는지 비파괴적 처리를 하는지 구분해 맞는 것에 O표

① [파괴적 처리 / 비파괴적 처리]

```
> const strA = "사과,배,바나나,귤"
undefined
> strA.split(",")
(4) ["사과", "배", "바나나", "귤"]
> strA
"사과,배,바나나,귤"
```

③ [파괴적 처리 / 비파괴적 처리]

```
> const arrayC = [1, 2, 3, 4, 5]
undefined
> arrayC.map((x) => x * x)
(5) [1, 4, 9, 16, 25]
> arrayC
(5) [1, 2, 3, 4, 5]
```

② [파괴적 처리 / 비파괴적 처리]

```
> const arrayB = ["사과", "배", "바나나",
"귤"]
undefined
> arrayB.push("감")
5
> arrayB
(5) ["사과", "배", "바나나", "귤", "감"]
```

④ [파괴적 처리 / 비파괴적 처리]

```
> const strD = " 여백이 포함된 메시지 "
undefined
> strD.trim()
"여백이 포함된 메시지"
> strD
" 여백이 포함된 메시지 "
```

SECTION 4-2 반복문(1)

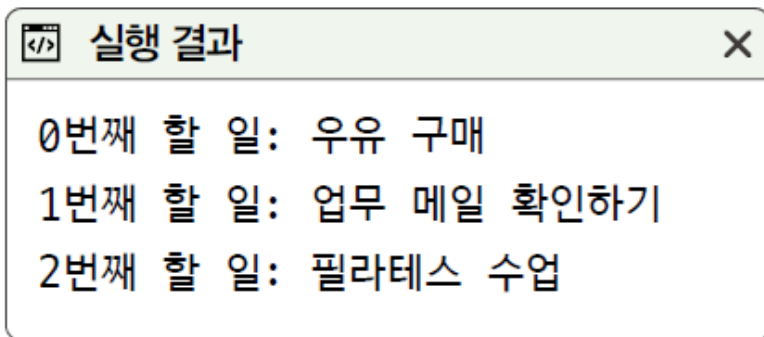
- for in 반복문

- 배열 요소를 하나하나 꺼내서 특정 문장을 실행할 때 사용

```
for (const 반복 변수 in 배열 또는 객체) {  
  문장  
}
```

- for in 반복문 (소스 코드 4-2-1.html)

```
01 <script>  
02  const todos = ['우유 구매', '업무 메일 확인하기', '필라테스 수업']  
03  
04  for (const i in todos) {  
05    console.log(`${i}번째 할 일: ${todos[i]}`)  
06  }  
07 </script>
```



※ for in 반복문은 구문 자체로 코드의 양이 어느 정도 있어서 코드를 하나하나 입력하는 것이 힘들 수 있음.
이럴 때 코드 블록을 사용. for를 입력하면 for와 관련된 여러 코드 블록이 나타나고 방향키를 사용해 for in 코드 블록으로 이동하고 Enter 또는 Tab 키를 클릭

SECTION 4-2 반복문(2)

◦ for of 반복문

- 요소의 값을 반복할 때 안정적으로 사용

```
for (const 반복 변수 of 배열 또는 객체) {  
  문장  
}
```

for in 반복문과 다르게 반복 변수에 요소의 값이 들어감

- for of 반복문 (소스 코드 4-2-2.html)

```
01 <script>  
02  const todos = ['우유 구매', '업무 메일 확인하기', '필라테스 수업']  
03  for (const todo of todos) {  
04    console.log(`오늘의 할 일: ${todo}`)  
05  }  
06 </script>
```

실행 결과

오늘의 할 일: 우유 구매
오늘의 할 일: 업무 메일 확인하기
오늘의 할 일: 필라테스 수업

SECTION 4-2 반복문(3)

◦ for 반복문

- 특정 횟수만큼 반복하고 싶을 때 사용하는 범용적인 반복문

```
for (let i = 0; i < 반복 횟수; i++) {  
  문장  
}
```

→ 다른 반복문과 다르게 반복 변수를 let 키워드로 선언합니다

- for 반복문 기본 (소스 코드 4-2-3.html)

```
01 <script>        
02 for (let i = 0; i < 5; i++) {  
03   console.log(`${i}번째 반복입니다.`)  
04 }  
05 </script>
```

→ 0부터 시작해서 5 미만이면 반복합니다. → 불 값

실행 결과



0번째 반복입니다.
1번째 반복입니다.
2번째 반복입니다.
3번째 반복입니다.
4번째 반복입니다.

SECTION 4-2 반복문(4)

- for 반복문
 - 1부터 N까지 더하기 (소스 코드 4-2-4.html)

```
01 <script>
02  let output = 0
03  for (let i = 1; i <= 100; i++) {
04    output += i
05  }
06  console.log(`1~100까지 숫자를 모두 더하면 ${output}입니다.`)
07 </script>
```

1부터 100까지 반복

 실행 결과 

1~100까지 숫자를 모두 더하면 5050입니다.

SECTION 4-2 반복문(5)

- for 반복문과 함께 배열 사용하기
 - for 반복문과 배열 (소스 코드 4-2-5.html)

```
01 <script>
02  const todos = ['우유 구매', '업무 메일 확인하기', '필라테스 수업']
03
04  for (let i = 0; i < todos.length; i++) {
05    console.log(`${i}번째 할 일: ${todos[i]}`)
06  }
07 </script>
```

실행 결과

0번째 할 일: 우유 구매
1번째 할 일: 업무 메일 확인하기
2번째 할 일: 필라테스 수업

- for 반복문으로 배열을 반대로 출력하기 (소스 코드 4-2-6.html)

```
01 <script>
02  const todos = ['우유 구매', '업무 메일 확인하기', '필라테스 수업']
03
04  for (let i = todos.length - 1; i >= 0; i--) {
05    console.log(`${i}번째 할 일: ${todos[i]}`)
06  }
07 </script>
```

실행 결과

2번째 할 일: 필라테스 수업
1번째 할 일: 업무 메일 확인하기
0번째 할 일: 우유 구매

배열의 마지막 요소부터 0까지 하나씩 빼면서 반복

SECTION 4-2 반복문(6)

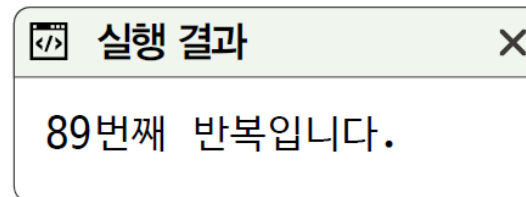
◦ while 반복문

- if 조건문과 다른 점은 문장을 한 번만 실행하고 끝나는 것이 아니라 불 표현식이 true면 계속해서 문장을 실행
- 조건이 변하지 않는다면 무한히 반복 실행하므로 조건을 거짓으로 만드는 내용이 문장에 포함되어야 함.
- 무한 루프: 반복문이 무한 반복되는 것

```
while (불 표현식) {  
    문장  
}
```

- 무한 반복문 (소스 코드 4-2-7.html)

```
01 <script>  
02  let i = 0  
03  while (true) {  
04    alert(`${i}번째 반복입니다.`)  
05    i = i + 1  
06  }  
07 </script>
```

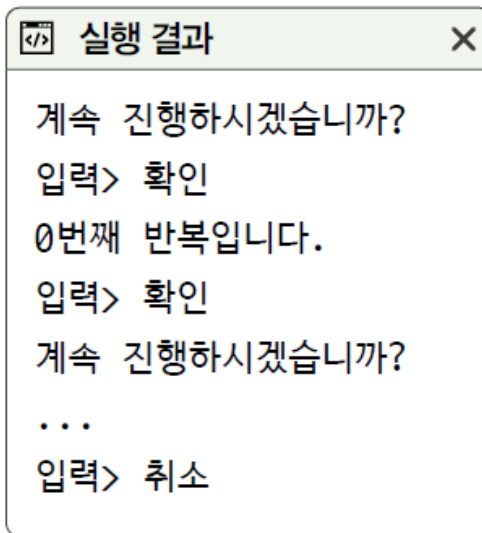


SECTION 4-2 반복문(7)

◦ while 반복문

- confirm() 함수를 사용하여 사용자에게 확인을 받는 대화상자 실행.
- [확인]은 true, [취소]는 false로 입력 받아 조건이 false(거짓)일 때 반복문 종료.
- While 반복문 기본 (소스 코드 4-2-8.html)

```
01 <script>
02 let i = 0
03 while (confirm('계속 진행하시겠습니까?')) {
04     // 사용자가 [확인] 버튼을 클릭하면 true가 되어 계속 반복합니다.
05     alert(`${i}번째 반복입니다.`)
06     i = i + 1
07 }
08 </script>
```



SECTION 4-2 반복문(8)

- while 반복문
 - while 반복문과 함께 배열 사용하기
 - 배열과 함께 사용하기 (소스 코드 4-2-9.html)

```
01 <script>
02 let i = 0
03 const array = [1, 2, 3, 4, 5]
04
05 while (i < array.length) {
06   console.log(`${i} : ${array[i]}`)
07   i++
08 }
09 </script>
```



```
실행 결과
0 : 1
1 : 2
2 : 3
3 : 4
4 : 5
```

※ 횟수를 기준으로 반복할 때는 코드를 간결하게 구현할 수 있는 for 반복문을 사용하는 것이 훨씬 편함.
while 반복문은 조건에 큰 비중이 있을 때 사용하는 것이 효과적
'특정 시간 동안 어떤 데이터를 받을 때까지', '배열에서 어떠한 요소가 완전히 제거될 때까지' 등 조건을
기반으로 사용하는 반복문에 while 반복문을 사용

SECTION 4-2 반복문(9)

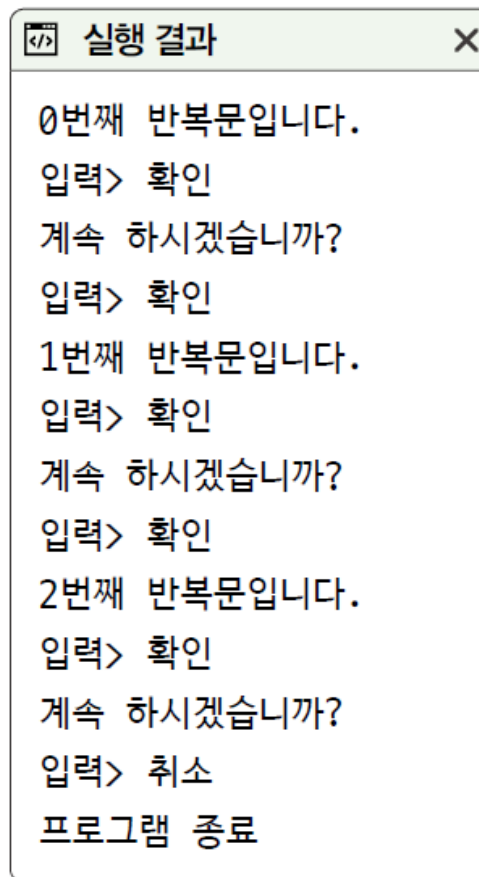
◦ break 키워드

- switch 조건문이나 반복문을 벗어날 때 사용하는 키워드

```
while (true) {  
    break  
}
```

- break 키워드 활용 (소스 코드 4-2-10.html)

```
01 <script>  
02 // 반복문  
03 for (let i = 0; true; i++) {  
04     alert(i + '번째 반복문입니다.')05  
06     // 진행 여부를 물어봅니다.  
07     const isContinue = confirm('계속 하시겠습니까?')  
08     if (!isContinue) {  
09         break  
10     }  
11 }  
12  
13 // 프로그램의 종료를 확인합니다.  
14 alert('프로그램 종료')  
15 </script>
```



SECTION 4-2 반복문(10)

- continue 키워드

- continue 키워드는 반복문 안의 반복 작업을 멈추고 반복문의 처음으로 돌아가 다음 반복 작업을 진행
- continue 키워드 활용(1) (소스 코드 4-2-11.html)

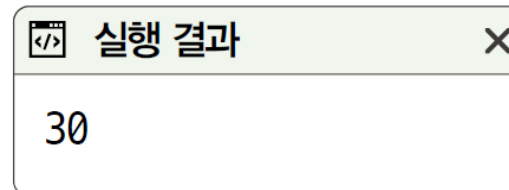
```
01 <script>
02  // 반복문
03  for (let i = 0; i < 5; i++) {
04    continue // 현재 반복 작업을 중지하고 다음 반복 작업을 수행합니다.
05    alert(i)  // 실행 안됨
06  }
07 </script>
```

SECTION 4-2 반복문(11)

- continue 키워드

- continue 키워드 활용(2) (소스 코드 4-2-12.html)

```
01 <script>
02  // 변수를 선언합니다.
03  let output = 0
04
05  // 반복문
06  for (let i = 1; i <= 10; i++) {
07    // 조건문
08    if (i % 2 === 1) {
09      // 홀수면 현재 반복을 중지하고 다음 반복을 수행합니다.
10      continue
11    }
12    output += i
13  }
14
15  // 출력합니다.
16  alert(output)
17 </script>
```



SECTION 4-2 반복문(12)

- 중첩 반복문을 사용하는 피라미드(누적 예제)
 - 중첩 반복문은 일반적으로 n-차원 처리를 할 때 사용
 - 중첩 반복문 사용하기(1)

```
*  
**  
***  
****  
*****  
*****  
*****  
*****  
*****
```

①

외부의 반복문
: 줄생성(\n)

l = 1

l = 2

l = 3

l = 4

② 내부의 반복문 : 별 생성(*)

* j = 0 → j < i(1) = 1번 반복해서 * 출력

** j = 0 → j < i(2) = 2번 반복해서 * 출력

*** j = 0 → j < i(3) = 3번 반복해서 * 출력

**** j = 0 → j < i(4) = 4번 반복해서 * 출력

SECTION 4-2 반복문(13)

- 중첩 반복문을 사용하는 피라미드(누적 예제)
 - 중첩 반복문 사용하기(1) (소스 코드 4-2-13.html)

```
01 <script>
02  // 변수를 선언합니다.
03  let output = ""
04
05  // 중첩 반복문
06  for (let i = 1; i < 10; i++) {
07    for (let j = 0; j < i; j++) {
08      output += '*'
09    }
10    output += '\n'
11  }
12
13  // 출력합니다.
14  console.log(output)
15 </script>
```

```
*
**
***
****
*****
*****
*****
*****
*****
*****
```

SECTION 4-2 반복문(14)

- 중첩 반복문을 사용하는 피라미드(누적 예제)
 - 중첩 반복문 사용하기(2)

[illegible]

외부의 반복문
: 줄생성(\n)

전체 높이
4로 가정

l = 1

l = 2

l = 3

l = 4

②

내부의 반복문
: 공백 생성

4개
3개
2개
1개

전체 높이
(4) -j개

③

내부의 두번째 반복문
: 별 생성(*)

□ □ □ □ *

□ □ □ ***

□ □ *****

□ *****

1개
3개
5개
7개

 $2k-1$ 개

SECTION 4-2 반복문(15)

- 중첩 반복문을 사용하는 피라미드(누적 예제)
 - 중첩 반복문 사용하기(2) (소스 코드 4-2-14.html)

```
01 <script>
02 // 변수를 선언합니다.
03 let output = ''
04
05 // 반복문
06 for (let i = 1; i < 15; i++) {
07   for (let j = 15; j > i; j--) {
08     output += ' '
09   }
10   for (let k = 0; k < 2 * i - 1; k++) {
11     output += '*'
12   }
13   output += '\n'
14 }
15
16 // 출력합니다.
17 console.log(output)
18 </script>
```

[마무리①]

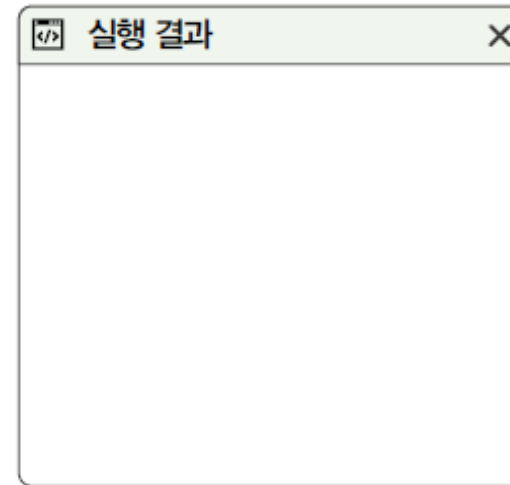
- 6가지 키워드로 정리하는 핵심 포인트
 - for in 반복문은 배열의 인덱스를 기반으로 반복할 때 사용
 - for of 반복문은 배열의 값을 기반으로 반복할 때 사용
 - for 반복문은 횟수를 기반으로 반복할 때 사용
 - while 반복문은 조건을 기반으로 반복할 때 사용
 - break 키워드는 switch 조건문이나 반복문을 벗어날 때 사용
 - continue 키워드는 반복문 안의 반복 작업을 멈추고 반복문의 처음으로 돌아가 다음 반복 작업을 진행

[마무리②]

- 확인 문제

- 다음 프로그램의 실행 결과를 예측해보기

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

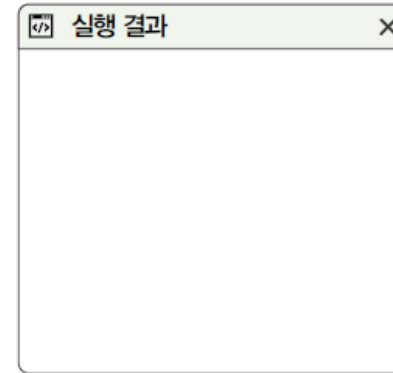


[마무리 ③]

◦ 확인 문제

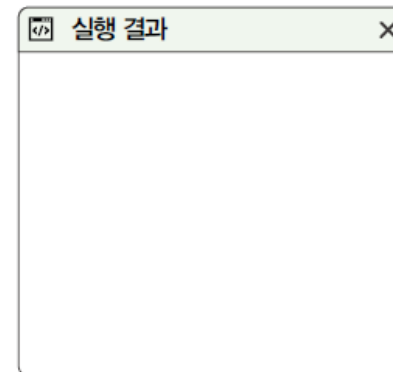
2. 다음 프로그램의 실행 결과를 예측해보고, 혹시 오류가 발생한다면 어디를 수정해야 하나?

```
<script>
  const array = []
  for (const i = 0; i < 3; i++) {
    array.push((i + 1) * 3)
  }
  console.log(array)
</script>
```



3. 1부터 100까지의 숫자를 곱한 값을 계산하는 프로그램을 만들고, 코드를 실행해 나온 결과를 확인하기

```
<script>
  let output = 1
  
  console.log(`1~100의 숫자를 모두 곱하면, ${output}입니다.`)
</script>
```



[마무리④]

◦ 확인 문제

4. 처음에는 조금 어려울 수 있겠지만, 활용 예제의 피라미드를 활용해서 다음과 같은 피라미드를 만들어보기

```
<script>  
  // 변수를 선언합니다.  
  let output = "  
  const size = 5  
  // 반복합니다.  
  
  
  
  
  
  
  
  
  // 출력합니다.  
  console.log(output)  
</script>
```

실행 결과

```
  *  
 ***  
*****  
*****  
*****  
*****  
*****  
*****  
***  
*
```