

# 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

- CHAPTER 06: 객체

SECTION 6-1 객체의 기본

SECTION 6-2 객체의 속성과 메소드 사용하기

SECTION 6-3 객체와 배열 고급



# CHAPTER 06 객체

객체의 속성과 메소드, 생성, 관리하는 기본 문법 학습

# SECTION 6-1 객체의 기본(1)

## ◦ 객체

- 배열은 요소에 접근할 때 인덱스를 사용하지만, 객체는 키(key)를 사용
- 객체는 중괄호{...}로 생성하며, 다음과 같은 형태의 자료를 쉼표(,)로 연결해서 입력

키: 값

- 객체 선언 예시

```
<script>
const product = {
  제품명: '7D 건조 망고',
  유형: '당절임',
  성분: '망고, 설탕, 메타중아황산나트륨, 치자황색소',
  원산지: '필리핀'
}
</script>
```

→ 키와 값 뒤에 쉼표(,)를  
넣어 구분

키	속성
제품명	7D 건조 망고
유형	당절임
성분	망고, 설탕, 메타중아황산나트륨, 치자황색소
원산지	필리핀

## SECTION 6-1 객체의 기본(2)

### ◦ 객체

- 객체 요소에 접근하기(대괄호 [ ] 사용)

---

product['제품명']	→ '7D 건조 망고'
product['유형']	→ '당절임'
product['성분']	→ '망고, 설탕, 메타중아황산나트륨, 치자황색소'
product['원산지']	→ '필리핀'

---

- 객체 요소에 접근하기(온점 . 사용)

---

product.제품명	→ '7D 건조 망고'
product.유형	→ '당절임'
product.성분	→ '망고, 설탕, 메타중아황산나트륨, 치자황색소'
product.원산지	→ '필리핀'

---

- 식별자로 사용할 수 없는 단어를 키로 사용할 경우
  - 객체를 생성할 때 키(key)는 식별자와 문자열을 모두 사용 가능  
대부분의 개발자가 식별자를 키로 사용하지만, 식별자로 사용할 수 없는 단어를 키로 사용할 때는 문자열을 사용
  - 식별자가 아닌 문자열을 키로 사용했을 때는 무조건 대괄호[...]를 사용해야 객체의 요소에 접근 가능

## SECTION 6-1 객체의 기본(3)

### ◦ 속성과 메소드

- 객체의 속성은 모든 형태의 자료값을 가질 수 있음
- 속성과 메소드 구분하기
  - 메소드: 객체의 속성 중 함수 자료형인 속성
  - eat 메소드

---

```
<script>
const pet = {
  name: '구름',
  eat: function (food) {}
}
// 메소드를 호출합니다.
person.eat()
</script>
```


---


## SECTION 6-1 객체의 기본(4)

### ◦ 속성과 메소드

- 메소드 내부에서 this 키워드 사용하기
  - 자기 자신의 자신이 가진 속성이라는 것을 표시할 때 this 키워드를 사용
- 메소드 내부에서의 this 키워드 (소스 코드 6-1-1.html)

```
01 <script>
02 // 변수를 선언합니다.
03 const pet = {
04   name: '구름',
05   eat: function (food) {
06     alert(this.name + '은/는 ' + food + '을/를 먹습니다.')
07   }
08 }
09
10 // 메소드를 호출합니다.
11 pet.eat('밥')
12 </script>
```

 this 키워드를 사용해 자신이 가진 속성에 접근할 수 있음

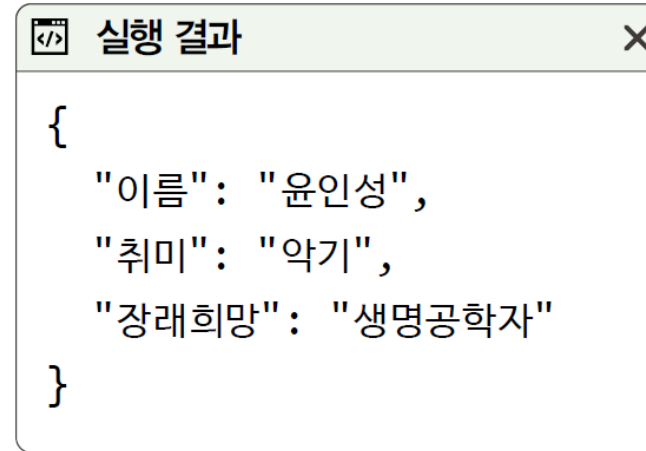
 실행 결과 ×

구름은/는 밥을/를 먹습니다.

## SECTION 6-1 객체의 기본(5)

- 동적으로 객체 속성 추가/제거
  - 동적으로 객체 속성 추가하기(소스 코드 6-1-2.html)

```
01 <script>
02  // 객체를 선언합니다.
03  const student = {}
04  student.이름 = '윤인성'
05  student.취미 = '악기'
06  student.장래희망 = '생명공학자'
07
08  // 출력합니다.
09  console.log(JSON.stringify(student, null, 2))
10 </script>
```



```
{
  "이름": "윤인성",
  "취미": "악기",
  "장래희망": "생명공학자"
}
```



## SECTION 6-1 객체의 기본(6)

- 동적으로 객체 속성 추가/제거

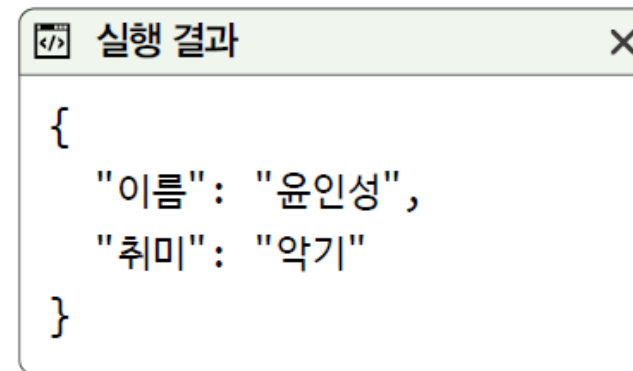
- 동적으로 객체 속성 제거하기

- delete 키워드 사용

delete 객체.속성

- 동적으로 객체 속성 제거하기 (소스 코드 6-1-3.html)

```
01 <script>
02  // 객체를 선언합니다.
03  const student = {}
04  student.이름 = '윤인성'
05  student.취미 = '악기'
06  student.장래희망 = '생명공학자'
07
08  // 객체의 속성을 제거합니다.
09  delete student.장래희망
10
11  // 출력합니다.
12  console.log(JSON.stringify(student, null, 2))
13 </script>
```



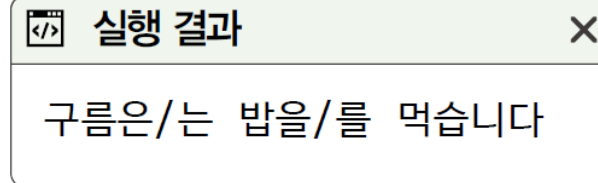
```
{
  "이름": "윤인성",
  "취미": "악기"
}
```

## SECTION 6-1 객체의 기본(7)

- 메소드 간단 선언 구문

- 메소드 선언 구문 (소스 코드 6-1-4.html)

```
01 <script>
02  // 변수를 선언합니다.
03  const pet = {
04    name: '구름',
05    eat (food) {
06      alert(this.name + '은/는 ' + food + '을/를 먹습니다.')
07    }
08  }
09
10  // 메소드를 호출합니다.
11  pet.eat('밥')
12 </script>
```

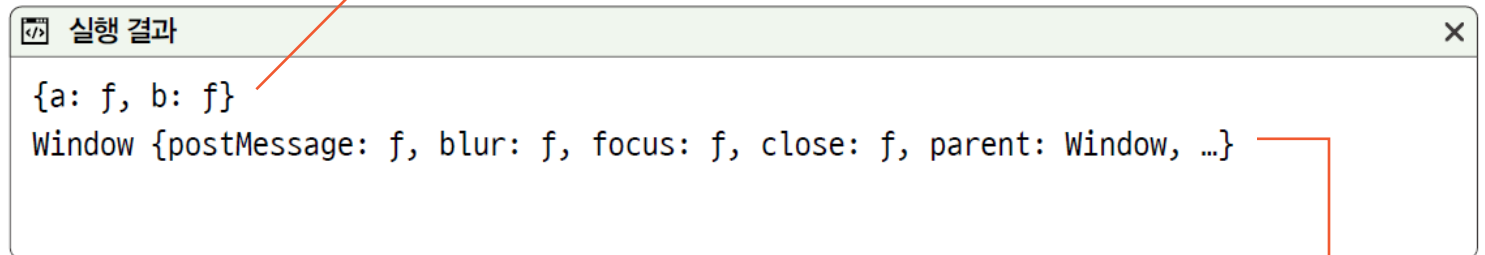


## [좀 더 알아보기] 화살표 함수를 사용한 메소드

- function () {} 형태로 선언하는 익명 함수와 () => {} 형태로 선언하는 화살표 함수는 객체의 메소드로 사용될 때 this 키워드를 다루는 방식이 다름
- this 키워드의 차이 (소스 코드 6-1-5.html)

```
01 <script>
02 // 변수를 선언합니다.
03 const test = {
04   a: function () { → 익명 함수로 선언
05     console.log(this)
06   },
07   b: () => { → 화살표 함수로 선언
08     console.log(this)
09   }
10 }
11
12 // 메소드를 호출합니다.
13 test.a()
14 test.b()
15 </script>
```

현재 코드에서 test 객체를 출력



window 객체를 출력

## [마무리①]

- 5가지 키워드로 정리하는 핵심 포인트
  - 요소란 배열 내부에 있는 값을 의미
  - 속성은 객체 내부에 있는 값을 의미
  - 메소드는 속성 중에 함수 자료형인 것을 의미
  - this 키워드는 객체 내부의 메소드에서 객체 자신을 나타내는 키워드
  - 객체 생성 이후에 속성을 추가하거나 제거하는 것을 동적 속성 추가, 동적 속성 제거라고 함

## [마무리②]

### ◦ 확인 문제

1. 다음과 같은 대상을 자바스크립트 객체로 선언하기. 자료형은 알맞다고 생각하는 것(문자열, 숫자, 불 등)으로 지정

속성 이름	속성 값
name	혼자 공부하는 파이썬
price	18000
publisher	한빛미디어

2. 다음 중 객체에 동적으로 속성을 추가하는 문법은?

① add 객체[속성] = 값    ② 객체.add('속성', 값)    ③ 객체[속성] = 값    ④ 객체[속성] add 값

3. 다음 중 객체에 동적으로 속성을 제거하는 문법은?

① delete 객체[속성]    ② 객체.delete('속성')    ③ delete 객체 from 속성    ④ delete 속성 from 객체

## [마무리 ③]

### ◦ 확인 문제

4. 다음 코드에서 메소드라고 부를 수 있는 속성에 동그라미 표시하고 코드의 실행 결과를 예측

```
<script>
const object = {
  ko: '빵',
  en: 'bread',
  ja: 'パン',
  fr: 'pain',
  es: 'pan',
  lang: {
    ko: '한국어',
    en: '영어',
    ja: '일본어',
    fr: '프랑스어',
    es: '스페인어'
  },
  print: function (lang) {
    console.log(`${this.ko}는 ${this.lang[lang]}로 ${this[lang]}입니다.`)
  }
}
object.print('es')
</script>
```



## SECTION 6-2 객체의 속성과 메소드 사용하기(1)

### ◦ 객체 자료형

- 속성과 메소드를 가질 수 있는 모든 것은 객체

- 배열도 객체, 함수도 객체

---

```
> const a = []  
undefined  
> a.sample = 10  
10  
> a.sample  
10
```

---

---

```
> function b () { }  
undefined  
> b.sample = 10  
10  
> b.sample  
10
```

---

- 배열인지 확인하려면 Array.isArray() 메소드를 사용 (Array도 메소드를 갖고 있으므로 객체)

---

```
> typeof a  
"object"  
> Array.isArray(a)  
true
```

---

- 함수는 '실행이 가능한 객체'. 자바스크립트에서는 함수를 일급 객체(first-class object) 또는 first-class citizen에 속한다고 표현

## SECTION 6-2 객체의 속성과 메소드 사용하기(2)

### ◦ 기본 자료형

- 기본 자료형(primitive types 또는 primitives): 실체가 있는 것(undefined와 null 등이 아닌 것) 중에 객체가 아닌 것
  - 숫자, 문자열, 불
  - 이러한 자료형은 객체가 아니므로 속성을 가질 수 없음

```
> const c = 273
undefined
> c.sample = 10
10
> c.sample
undefined
```

속성을 만들 수 있는 것처럼 보이지만  
실제로 속성이 만들어지지 않음

```
> const d = '안녕하세요'
undefined
> d.sample = 10
10
> d.sample
undefined
> const e = true
undefined
> e.sample = 10
10
> e.sample
undefined
```

속성이 추가되지 않음



## SECTION 6-2 객체의 속성과 메소드 사용하기(3)

### ◦ 기본 자료형을 객체로 선언하기

- 숫자 객체, 문자열 객체, 불 객체를 생성
  - 단순한 기본 자료형이 아니므로 이전과 다르게 속성을 가짐

---

```
const 객체 = new 객체 자료형 이름()
```

---



---

```
new Number(10)  
new String('안녕하세요')  
new Boolean(true)
```

---

---

기본편 260 Chapter 06 | 객체

```
> const f = new Number(273)
```

```
undefined
```

```
> typeof f
```

```
"object"
```

```
> f.sample = 10
```

```
10
```

```
> f.sample
```

```
10
```

```
> f
```

```
Number {273, sample: 10}
```

```
> f + 0
```

```
273
```

```
> f.valueOf()
```

```
273
```

---

속성을 가질 수 있음

콘솔에서 단순히 f를 출력하면 객체 형태로 출력

숫자와 똑같이 활용할 수 있고 valueOf() 메소드를 사용해서 값을 추출할 수도 있음

## SECTION 6-2 객체의 속성과 메소드 사용하기(4)

### ◦ 기본 자료형의 일시적 승급

- 자바스크립트는 사용의 편리성을 위해서 기본 자료형의 속성과 메소드를 호출할 때(기본 자료형 뒤에 온점(.)을 찍고 무언가 하려고 하면) 일시적으로 기본 자료형을 객체로 승급시킴

---

```
> const h = '안녕하세요'
```

```
undefined
```

```
> h.sample = 10
```

```
10
```

```
> h.sample
```

```
undefined
```

---

→ 일시적으로 객체로 승급되어 sample 속성을 추가할 수 있음

→ 일시적으로 승급된 것이라 추가했던 sample 속성은 이미 사라짐

## SECTION 6-2 객체의 속성과 메소드 사용하기(5)

### 프로토타입으로 메소드 추가하기

- prototype 객체에 속성과 메소드를 추가하면 모든 객체(와 기본 자료형)에서 해당 속성과 메소드를 사용할 수 있음

---

```
객체 자료형 이름.prototype.메소드 이름 = function () {  
}
```

---

- 프로토타입으로 숫자 메소드 추가하기 (소스 코드 6-2-1.html)

---

```
01 <script>  
02 // power() 메소드를 추가합니다.  
03 Number.prototype.power = function (n = 2) {  
04   return this.valueOf() ** n  
05 }  
06  
07 // Number 객체의 power() 메소드를 사용합니다.  
08  
09 const a = 12  
10 console.log('a.power():', a.power())  
11 console.log('a.power(3):', a.power(3))  
12 console.log('a.power(4):', a.power(4))  
13 </script>
```

---

실행 결과

```
a.power(): 144  
a.power(3): 1728  
a.power(4): 20736
```

## SECTION 6-2 객체의 속성과 메소드 사용하기(6)

### ◦ 프로토타입으로 메소드 추가하기

- indexOf() 메소드로 자바스크립트에서 문자열 내부에 어떤 문자열이 있는지, 배열 내부에 어떤 자료가 있는지 확인
  - 문자열 '안녕하세요' 내부에 '안녕', '하세', '없는 문자열'이 있는지 확인하면, 해당 문자열이 시작하는 위치(인덱스)를 출력하고, 없으면 -1을 출력

---

```
> const j = '안녕하세요'
```

```
undefined
```

```
> j.indexOf('안녕')
```

```
0
```

```
> j.indexOf('하세')
```

```
2
```

```
> j.indexOf('없는 문자열')
```

```
-1
```

---

→ 문자열 내에 있는 문자열이라면 그 인덱스를 출력

→ 문자열 내에 없는 문자열이라면 -1을 출력

- 배열의 indexOf() 메소드도 마찬가지로 작동

## SECTION 6-2 객체의 속성과 메소드 사용하기(7)

- 프로토타입으로 메소드 추가하기
  - 프로토타입으로 문자열 메소드 추가하기 (소스 코드 6-2-2.html)

```
01 <script>
02 // contain() 메소드를 추가합니다.
03 String.prototype.contain = function (data) {
04     return this.indexOf(data) >= 0
05 }
06
07 Array.prototype.contain = function (data) {
08     return this.indexOf(data) >= 0
09 }
10
11 // String 객체의 contain() 메소드를 사용합니다.
12 const a = '안녕하세요'
13 console.log('안녕 in 안녕하세요:', a.contain('안녕'))
14 console.log('없는데 in 안녕하세요:', a.contain('없는데'))
15
16 // Array 객체의 contain() 메소드를 사용합니다.
17 const b = [273, 32, 103, 57, 52]
18 console.log('273 in [273, 32, 103, 57, 52]:', b.contain(273))
19 console.log('0 in [273, 32, 103, 57, 52]:', b.contain(0))
20 </script>
```

실행 결과

```
안녕 in 안녕하세요: true
없는데 in 안녕하세요: false
273 in [273, 32, 103, 57, 52]: true
0 in [273, 32, 103, 57, 52]: false
```

## SECTION 6-2 객체의 속성과 메소드 사용하기(8)

- Number 객체

- 숫자 N번째 자릿수까지 출력하기: toFixed()

- toFixed() 메소드는 소수점 이하 몇 자리까지만 출력하고 싶을 때 사용

---

```
> const l = 123.456789
```

```
undefined
```

```
> l.toFixed(2)
```

```
"123.46"
```

```
> l.toFixed(3)
```

```
"123.457"
```

```
> l.toFixed(4)
```

```
"123.4568"
```

---

- NaN과 Infinity 확인하기: isNaN(), isFinite()

- Number 뒤에 점을 찍고 사용

## SECTION 6-2 객체의 속성과 메소드 사용하기(9)

### ◦ Number 객체

- NaN과 Infinity 확인하기: isNaN(), isFinite()
  - Number 뒤에 점을 찍고 사용

isNaN()

```
> m
NaN
> m === NaN ] NaN을 생성
false
> Number.isNaN(m) ] NaN과 비교해서는
true               ] NaN인지 확인 불가
```

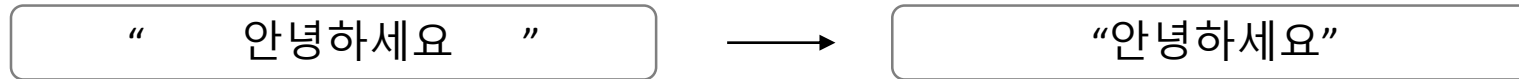
isFinite()

```
> const n = 10 / 0
undefined
> n
Infinity → 양의 무한대를 생성
> const o = -10 / 0
undefined
> o
-Infinity → 음의 무한대를 생성
> Number.isFinite(n) ]
false                ] isFinite(유한한 숫자인가?)가 false로 나옴
> Number.isFinite(o) ]
false
> Number.isFinite(1) ]
true                  ] 일반적인 숫자는 셀 수 있으므로 true가 나옴
> Number.isFinite(10) ]
true
```

## SECTION 6-2 객체의 속성과 메소드 사용하기(10)

### ◦ String 객체

- 문자열 양쪽 끝의 공백 없애기: trim()



```
> const stringA = `
메시지를 입력하다보니 앞에 줄바꿈도 들어가고`
undefined
> const stringB = ` 앞과 뒤에 공백도 들어가고 `
Undefined
> stringA
"
메시지를 입력하다보니 앞에 줄바꿈도 들어가고"
> stringB
" 앞과 뒤에 공백도 들어가고 "
> stringA.trim()
"메시지를 입력하다보니 앞에 줄바꿈도 들어가고"
> stringB.trim()
"앞과 뒤에 공백도 들어가고"
```

문자열 앞뒤 공백이 제거

[참조] 모질라 String 객체의 속성과 메소드

[https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global\\_Objects/String](https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/String)



## SECTION 6-2 객체의 속성과 메소드 사용하기(11)

### ◦ String 객체

#### - 문자열을 특정 기호로 자르기: split()

- split() 메소드는 문자열을 매개변수(다른 문자열)로 잘라서 배열을 만들어 리턴하는 메소드

```
> input = input.trim()
"일자,달러,엔,유로
02,1141.8,1097.46,1262.37
03,1148.7,1111.36,1274.65
04,1140.6,1107.81,1266.58
07,1143.4,1099.58,1267.8
08,1141.6,1091.97,1261.07"
```

→ 앞뒤 공백을 제거

```
> input = input.split('\n')
["일자,달러,엔,유로", "02,1141.8,1097.46,1262.37",
"03,1148.7,1111.36,1274.65",
"04,1140.6,1107.81,1266.58", "07,1143.4,1099.58,1267.8",
"08,1141.6,1091.97,1261.07"]
```

→ 줄바꿈으로 자르기

```
> input = input.map((line) => line.split(','))
[Array(4), Array(4), Array(4), Array(4), Array(4), Array(4)]
> JSON.stringify(input, null, 2)
```

→ 배열 내부의 문자열들을 쉼표로 자르기

```
"[
[
"일자",
"달러",
"엔",
"유로"
],
```

이하 생략

## SECTION 6-2 객체의 속성과 메소드 사용하기(12)

### ◦ JSON 객체

- 인터넷에서 문자열로 데이터를 주고 받을 때는 CSV, XML, CSON 등의 다양한 자료 표현 방식을 사용. 현재 가장 많이 사용되는 자료 표현 방식은 JSON 객체
  - 값을 표현할 때는 문자열, 숫자, 불 자료형만 사용할 수 있음(함수 등은 사용 불가).
  - 문자열은 반드시 큰따옴표로 만들어야 함
  - 키key에도 반드시 따옴표를 붙여야 함
- JSON을 사용하여 '책'을 표현한 예
  - 하나의 자료

---

```
{  
  "name": "혼자 공부하는 파이썬",  
  "price": 18000,  
  "publisher": "한빛미디어"  
}
```

---

- 여러 개의 자료

---

```
[{  
  "name": "혼자 공부하는 파이썬",  
  "price": 18000,  
  "publisher": "한빛미디어"  
}, {  
  "name": "HTML5 웹 프로그래밍 입문",  
  "price": 26000,  
  "publisher": "한빛아카데미"  
}]
```

---

## SECTION 6-2 객체의 속성과 메소드 사용하기(13)

### ◦ JSON 객체

- 자바스크립트 객체를 JSON 문자열로 변환할 때는 JSON.stringify() 메소드를 사용
- JSON.stringify() 메소드 (소스 코드 6-2-3.html)

```
01 <script>
02 // 자료를 생성합니다.
03 const data = [{
04   name: '혼자 공부하는 파이썬',
05   price: 18000,
06   publisher: '한빛미디어'
07 }, {
08   name: 'HTML5 웹 프로그래밍 입문',
09   price: 26000,
10   publisher: '한빛아카데미'
11 }]
12
13 // 자료를 JSON으로 변환합니다.
14 console.log(JSON.stringify(data))
15 console.log(JSON.stringify(data, null, 2))
16 </script>
```

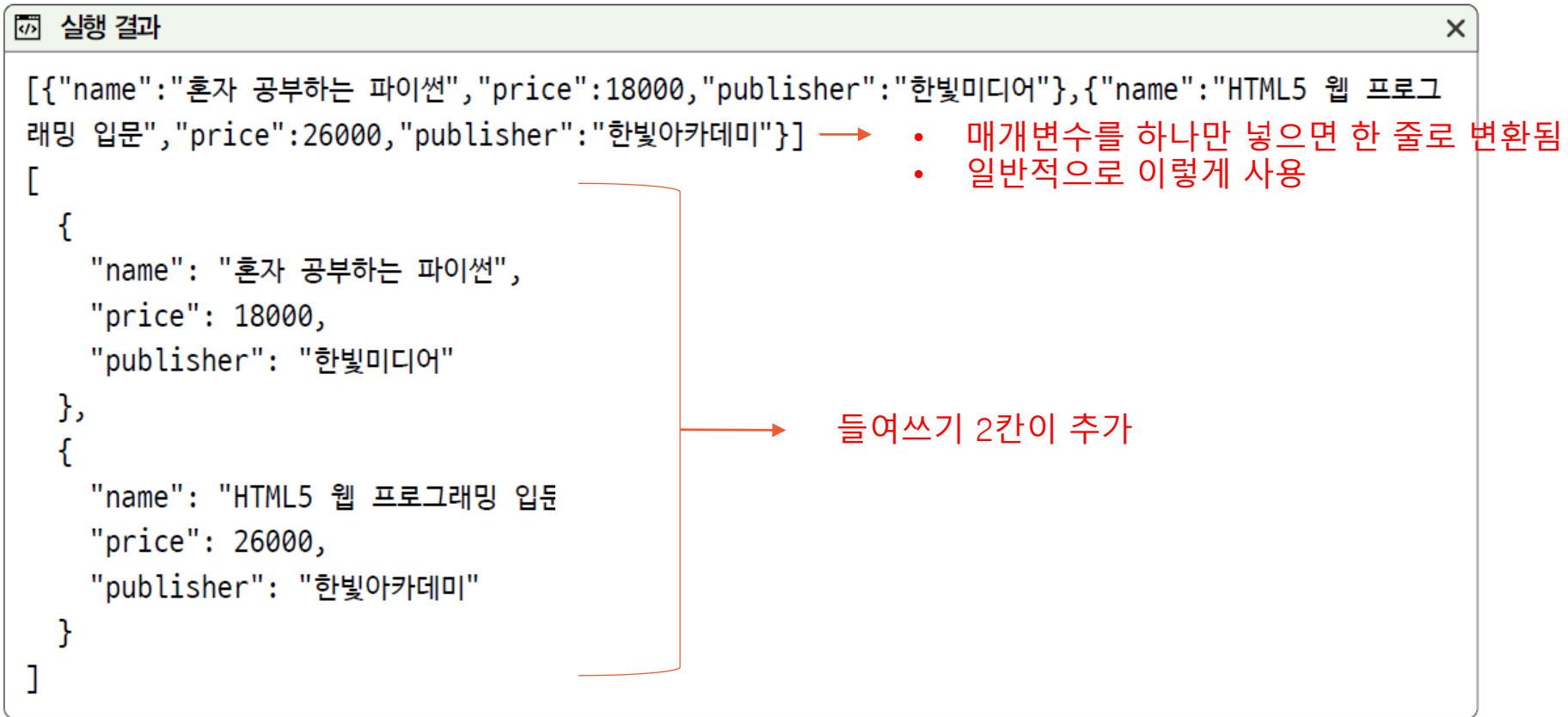
2번째 매개변수는 객체에서 어떤 속성만 선택해서 추출하고 싶을 때 사용하나 거의 사용하지 않으며, 일반적으로 null(아무 것도 없음)을 넣음

들여쓰기 2칸으로 설정

## SECTION 6-2 객체의 속성과 메소드 사용하기(14)

### ◦ JSON 객체

- 자바스크립트 객체를 JSON 문자열로 변환할 때는 JSON.stringify() 메소드를 사용
- JSON.stringify() 메소드를 출력한 결과 (소스 코드 6-2-3.html)



```
[{"name": "혼자 공부하는 파이썬", "price": 18000, "publisher": "한빛미디어"}, {"name": "HTML5 웹 프로그래밍 입문", "price": 26000, "publisher": "한빛아카데미"}]
```

→

- 매개변수를 하나만 넣으면 한 줄로 변환됨
- 일반적으로 이렇게 사용

↓

```
[  
  {  
    "name": "혼자 공부하는 파이썬",  
    "price": 18000,  
    "publisher": "한빛미디어"  
  },  
  {  
    "name": "HTML5 웹 프로그래밍 입문",  
    "price": 26000,  
    "publisher": "한빛아카데미"  
  }  
]
```

→ 들여쓰기 2칸이 추가

## SECTION 6-2 객체의 속성과 메소드 사용하기(15)

### ◦ JSON 객체

- JSON 문자열을 자바스크립트 객체로 전개할 때는 JSON.parse() 메소드를 사용
- JSON.parse() 메소드 소스 코드 6-2-4.html)

```
01 <script>
02 // 자료를 생성합니다.
03 const data = [{
04   name: '혼자 공부하는 파이썬',
05   price: 18000,
06   publisher: '한빛미디어'
07 }, {
08   name: 'HTML5 웹 프로그래밍 입문',
09   price: 26000,
10   publisher: '한빛아카데미'
11 }]
12
13 // 자료를 JSON으로 변환합니다.
14 const json = JSON.stringify(data)
15 console.log(json)
16
17 // JSON 문자열을 다시 자바스크립트 객체로 변환합니다.
18 console.log(JSON.parse(json))
```

실행 결과

```
[{"name":"혼자 공부하는 파이썬","price":18000,"publisher":"한빛미디어"}, {"name":"HTML5 웹 프로그래밍 입문","price":26000,"publisher":"한빛아카데미"}]

Array(2)
  0: {name: '혼자 공부하는 파이썬', price: 18000, publisher: '한빛미디어'}
  1: {name: "HTML5 웹 프로그래밍 입문", price: 26000, publisher: "한빛아카데미"}
  length: 2
  __proto__: Array(0)
```

## SECTION 6-2 객체의 속성과 메소드 사용하기(16)

- Math 객체

- 수학과 관련된 기본적인 연산을 할 때는 Math 객체를 사용
  - Math 객체 속성으로는 pi, e와 같은 수학 상수가 있음
  - 메소드로는 Math.sin(), Math.cos(), Math.tan()와 같은 삼각함수도 있음
  - 랜덤한 숫자를 생성할 때 사용되는 Math.random() 메소드는 0이상, 1 미만의 랜덤한 숫자를 생성
- [참조] 모질라 Math 객체의 속성과 메소드
  - URL [https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global\\_Objects/Math](https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/Math)

## SECTION 6-2 객체의 속성과 메소드 사용하기(17)

- Math 객체

- Math.random() 메소드 (소스 코드 6-2-5.html)

---

```
01 <script>
02  const num = Math.random()
03
04  console.log('# 랜덤한 숫자')
05  console.log('0-1 사이의 랜덤한 숫자:', num) → 0 <= 결과 < 1의 범위를 가짐
06  console.log("")
07
08  console.log('# 랜덤한 숫자 범위 확대')
09  console.log('0~10 사이의 랜덤한 숫자:', num * 10) → 0 <= 결과 < 10의 범위를 가짐
10  console.log('0~50 사이의 랜덤한 숫자:', num * 50)
11  console.log("")
12
13  console.log('# 랜덤한 숫자 범위 이동')
14  console.log('-5~5 사이의 랜덤한 숫자:', num * 10 - 5) → -5 <= 결과 < 5의 범위를 가짐
15  console.log('-25~25 사이의 랜덤한 숫자:', num * 50 - 25)
16  console.log("")
17
18  console.log('# 랜덤한 정수 숫자')
19  console.log('-5~5 사이의 랜덤한 정수 숫자:', Math.floor(num * 10 - 5))
20  console.log('-25~25 사이의 랜덤한 정수 숫자:', Math.floor(num * 50 - 25))
21 </script>
```

---

## SECTION 6-2 객체의 속성과 메소드 사용하기(18)

### ◦ Math 객체

- Math.random() 메소드 실행 결과(소스 코드 6-2-5.html)
  - 코드를 실행할 때마다 랜덤한 숫자가 다르므로 결과 역시 다르게 나옴

```
실행 결과(1)
# 랜덤한 숫자
0~1 사이의 랜덤한 숫자: 0.07432212812757388

# 랜덤한 숫자 범위 확대
0~10 사이의 랜덤한 숫자: 0.7432212812757388
0~50 사이의 랜덤한 숫자: 3.716106406378694

# 랜덤한 숫자 범위 이동
-5~5 사이의 랜덤한 숫자: -4.256778718724261
-25~25 사이의 랜덤한 숫자: -21.2838935936213

# 랜덤한 정수 숫자
-5~5 사이의 랜덤한 정수 숫자: -5
-25~25 사이의 랜덤한 정수 숫자: -22
```

```
실행 결과(2)
# 랜덤한 숫자
0~1 사이의 랜덤한 숫자: 0.6780090022598715

# 랜덤한 숫자 범위 확대
0~10 사이의 랜덤한 숫자: 6.780090022598715
0~50 사이의 랜덤한 숫자: 33.900450112993575

# 랜덤한 숫자 범위 이동
-5~5 사이의 랜덤한 숫자: 1.7800900225987153
-25~25 사이의 랜덤한 숫자: 8.900450112993575

# 랜덤한 정수 숫자
-5~5 사이의 랜덤한 정수 숫자: 1
-25~25 사이의 랜덤한 정수 숫자: 8
```



## SECTION 6-2 객체의 속성과 메소드 사용하기(19)

- 외부 script 파일 읽어들이기
  - 프로그램의 규모가 커지면 파일 하나가 너무 방대해지므로 파일을 분리할 필요가 있음
  - 별도의 자바스크립트 파일을 만들기 위해, 비주얼 스튜디오 코드에서 main.html과 test.js라는 이름으로 파일을 생성해서 같은 폴더에 저장하기
  - 외부 자바스크립트 파일을 읽어들이는 때도 script 태그를 사용

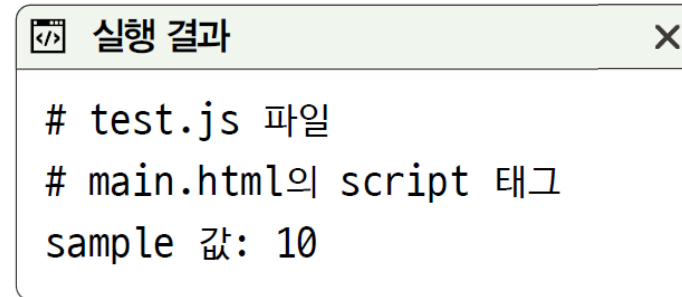
## SECTION 6-2 객체의 속성과 메소드 사용하기(20)

- 외부 script 파일 읽어들이기
  - 외부 script 파일 읽어들이기(1) (소스 코드 main.html)

```
01 <!DOCTYPE html>
02 <html>
03 <head>
04   <title></title>
05   <script src="test.js"></script>
06   <script>
07     console.log('# main.html의 script 태그')
08     console.log('sample 값:', sample)
09   </script>
10 </head>
11 <body>
12
13 </body>
14 </html>
```

- 외부 script 파일 읽어들이기(2) 소스 코드 test.js

```
01 console.log('# test.js 파일')
02 const sample = 10
```



main.html 파일에서 5행의 외부 자바스크립트를 읽어들이는 script 태그(<script src="test.js"></ script>)가 6~9행의 코드가 적혀 있는 script 태그보다 위에 있으므로 먼저 실행

## SECTION 6-2 객체의 속성과 메소드 사용하기(21)

### ◦ Lodash 라이브러리

- 개발할 때 보조적으로 사용하는 함수들을 제공하는 유틸리티 라이브러리 중 가장 많이 사용
  - lodash 라이브러리 다운로드 페이지  
<https://lodash.com>
  - Lodash CDN 링크 페이지  
<https://www.jsdelivr.com/package/npm/lodash>
- CDN(Contents Delivery Network)은 콘텐츠 전송 네트워크
- min 버전: 자바스크립트 코드를 집핑(zipping)한 파일
  - 집핑(zipping): 데이터를 CDN으로 전송하는 경우 데이터의 용량을 줄이고자 다음과 같이 소개를 줄이고 모든 코드를 응축
- 다양한 Lodash 라이브러리
  - Luxon와 date-fns: 날짜와 시간을 쉽게 다루는 라이브러리
  - Handsontable: 웹 페이지에 스프레드시트를 출력하는 라이브러리
  - D3.js와 ChartJS: 그래프를 그릴 수 있는 라이브러리
  - Three.js: 3차원 그래픽을 다루는 라이브러리

## SECTION 6-2 객체의 속성과 메소드 사용하기(22)

### ◦ Lodash 라이브러리

- sortBy() 메소드: 배열을 어떤 것으로 정렬할지 지정하면, 지정한 것을 기반으로 배열을 정렬해서 리턴
- sortBy() 메소드 (소스 코드 6-2-6.html)

```
01 <script src="https://cdn.jsdelivr.net/npm/lodash@4.17.15/lodash.min.js">
02 </script>
03 <script>
04 // 데이터를 생성합니다.
05 const books = [{
06   name: '혼자 공부하는 파이썬',
07   price: 18000,
08   publisher: '한빛미디어'
09 }, {
10   name: 'HTML5 웹 프로그래밍 입문',
11   price: 26000,
12   publisher: '한빛아카데미'
13 }, {
```

중간 생략

```
21 }]
22
23 // 가격으로 정렬한 뒤 출력합니다.
24 const output = _.sortBy(books, (book) => book.price)
25 console.log(JSON.stringify(output, null, 2))
26 </script>
```

실행 결과

```
[
  {
    "name": "혼자 공부하는 파이썬",
    "price": 18000,
    "publisher": "한빛미디어"
  },
  {
    "name": "딥러닝을 위한 수학",
    "price": 25000,
    "publisher": "위키북스"
  },
  {
    "name": "HTML5 웹 프로그래밍 입문",
    "price": 26000,
    "publisher": "한빛아카데미"
  },
  {
    "name": "머신러닝 딥러닝 실전 개발 입문",
    "price": 30000,
    "publisher": "위키북스"
  }
]
```

## [마무리①]

### ◦ 4가지 키워드로 정리하는 핵심 포인트

- 실체가 있는 것 중에서 객체가 아닌 것을 기본 자료형이라고 하며, 숫자, 문자열, 불이 대표적인 예
- 객체를 기반으로 하는 자료형을 객체 자료형이라고 하며, new 키워드를 활용해서 생성
- 기본 자료형의 승급이란 기본 자료형이 일시적으로 객체 자료형으로 변화하는 것을 의미
- prototype 객체란 객체의 틀을 의미하며, 이곳에 속성과 메소드를 추가하면 해당 객체 전체에서 사용

### ◦ 확인 문제

1. 다음 코드의 실행 결과를 예측해보세요. 예측과 다른 결과가 나온다면 왜 그런지 생각해보기

```
<script>
  const num = 52000
  num.원 = function () {
    return this.valueOf() + '원'
  }
  console.log(num.원())
</script>
```

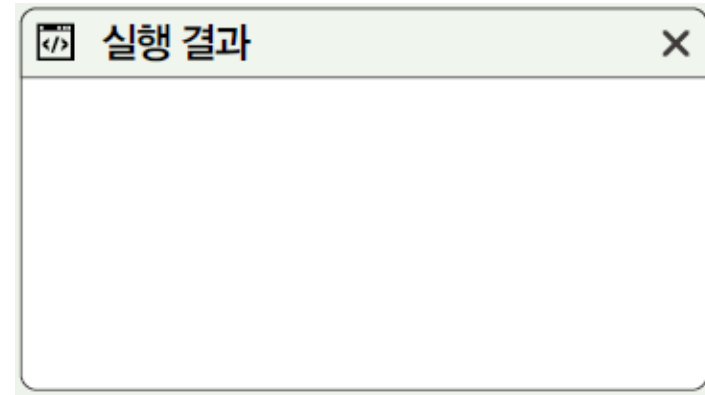


## [마무리②]

- 확인 문제

- 2. 다음 코드의 실행 결과를 예측하기

```
<script>
function printLang(code) {
  return printLang._lang[code]
}
printLang._lang = {
  ko: '한국어',
  en: '영어',
  ja: '일본어',
  fr: '프랑스어',
  es: '스페인어'
}
console.log('printLang("ko"):', printLang('ko'))
console.log('printLang("en"):', printLang('en'))
</script>
```



## [마무리③]

### ◦ 확인 문제

3. 모질라 문서에서 Math 객체와 관련된 내용을 읽고 사인 90도의 값을 구하기. 참고로 사인 90도는 1. 아주 단순하게 생각해서 구현하면 0.8939966636005579라는 결과가 나옴. 0.8939966636005579가 나왔다면 왜 그런지, 그리고 이를 어떻게 해야 제대로 사용할 수 있는지 구글 검색 등을 활용해서 알아보고 코드를 수정해보기

```
<script>  
  // 변수를 선언합니다.  
  const degree = 90  
  // 출력합니다.  
    
</script>
```

4. 다음 중 어떤 종류의 객체들이 모두 공유하는 속성과 메소드를 추가할 때 사용하는 객체의 이름은?

① classProp      ② prototype      ③ sample      ④ frame

## [마무리④]

### ◦ 확인 문제

5. 본문에서는 Lodash 라이브러리의 `_.sortBy()` 메소드를 살펴보았음. `_.orderBy()` 메소드도 한번 살펴보고 어떤 형태로 사용해야 하는지 직접 예제를 작성하기. 그리고 다음과 같은 배열을 이름(name)으로 오름차순 정렬하기

```
<script>
const books = [{
  name: '혼자 공부하는 파이썬',
  price: 18000,
  publisher: '한빛미디어'
}, {
  name: 'HTML5 웹 프로그래밍 입문',
  price: 26000,
  publisher: '한빛아카데미'
}, {
  name: '머신러닝 딥러닝 실전 개발 입문',
  price: 30000,
  publisher: '위키북스'
}, {
  name: '딥러닝을 위한 수학',
  price: 25000,
  publisher: '위키북스'
}]
</script>
```

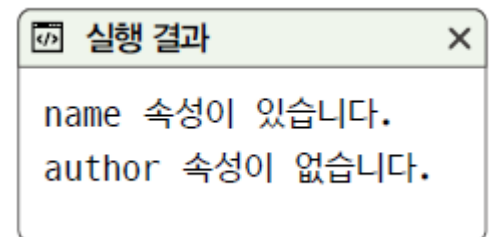


## SECTION 6-3 객체와 배열 고급(1)

### ◦ 속성 존재 여부 확인

- 객체 내부에 어떤 속성이 있는지 확인하는 코드는 굉장히 자주 사용. 내가 직접 코드는 물론 남이 만든 코드를 이해할 때도 필요.
- 조건문으로 undefined인지 아닌지 확인하면 속성 존재 여부를 확인할 수 있음.
- 속성 존재 여부 확인하기 (소스 코드 6-3-1.html)

```
01 <script>
02  // 객체를 생성합니다.
03  const object = {
04    name: '혼자 공부하는 파이썬',
05    price: 18000,
06    publisher: '한빛미디어'
07  }
08
09  // 객체 내부에 속성이 있는지 확인합니다.
10  if (object.name !== undefined) {
11    console.log('name 속성이 있습니다.')
12  } else {
13    console.log('name 속성이 없습니다.')
14  }
15
16  if (object.author !== undefined) {
17    console.log('author 속성이 있습니다.')
18  } else {
19    console.log('author 속성이 없습니다.')
20  }
21 </script>
```



## SECTION 6-3 객체와 배열 고급(2)

- 개발자들은 일반적으로 더 간단하게 검사하려고 다음과 같이 사용하기도 함.
- 단, 객체의 특정 속성이 false로 변환될 수 있는 값(0, false, 빈 문자열 등)이 아닐 때와 같은 전제가 있어야 안전하게 사용할 수 있음.

---

// 객체 내부에 속성이 있는지 확인합니다.

```
if (object.name) {  
  console.log('name 속성이 있습니다.')  
} else {  
  console.log('name 속성이 없습니다.')  
}  
  
if (object.author) {  
  console.log('author 속성이 있습니다.')  
} else {  
  console.log('author 속성이 없습니다.')  
}
```

---

→ 짧은 조건문으로 더 짧게도 사용 가능

---

// 객체 내부에 속성이 있는지 확인합니다.

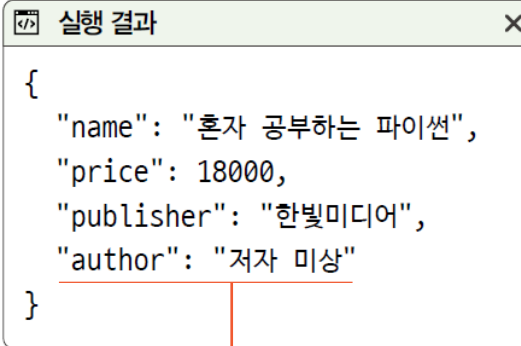
```
object.name || console.log('name 속성이 없습니다.')  
object.author || console.log('author 속성이 없습니다.')
```

---

## SECTION 6-3 객체와 배열 고급(3)

- 이러한 조건문을 활용해서 객체의 기본 속성을 지정하는 경우도 많음
- 다음은 객체의 속성이 있는지 확인하고 있다면 해당 속성을, 없다면 별도의 문자열을 지정하는 코드
- 기본 속성 지정하기 (소스 코드 6-3-2.html)

```
01 <script>
02  // 객체를 생성합니다.
03  const object = {
04    name: '혼자 공부하는 파이썬',
05    price: 18000,
06    publisher: '한빛미디어'
07  }
08
09  // 객체의 기본 속성을 지정합니다.
10  object.name = object.name !== undefined ? object.name : '제목 미정'
11  object.author = object.author !== undefined ? object.author : '저자 미상'
12
13  // 객체를 출력합니다.
14  console.log(JSON.stringify(object, null, 2))
15 </script>
```



```
{
  "name": "혼자 공부하는 파이썬",
  "price": 18000,
  "publisher": "한빛미디어",
  "author": "저자 미상"
}
```

author 속성이 없었으므로  
기본 속성이 적용됨

- 마찬가지로 속성이 false로 변환될 수 있는 값이 들어오지 않을 것이라는 전제가 있으면 짧은 조건문으로도 구현

```
// 객체의 기본 속성을 지정합니다.
object.name = object.name || '제목 미정'
object.author = object.author || '저자 미상'
```

## SECTION 6-3 객체와 배열 고급(4)

### 배열 기반의 다중 할당

- 최신 자바스크립트부터 배열과 비슷한 작성 방법으로 한 번에 여러 개의 변수에 값을 할당하는 다중 할당 기능이 추가.

---

[식별자, 식별자, 식별자, ...] = 배열

---

- 할당 연산자(=) 왼쪽에 식별자(변수 또는 상수)의 배열을 넣고, 오른쪽에 배열을 위치시키면 배열의 위치에 맞게 값들이 할당.
  - 처음에 [a, b] = [1, 2]라고 할당했으므로 a에 1이 할당, b에 2가 할당.
  - 이때 let [a, b] 형태로 선언했으므로 a와 b는 변수가 됨.
  - 배열의 크기는 값을 필요도 없고 const 키워드로도 사용할 수 있음.
- 
- 오른쪽 같이 배열의 길이가 5인 arrayA의 값을 [a, b, c]에 할당하면 앞의 3개만 할당됨.

```
> let [a, b] = [1, 2]
undefined
> console.log(a, b)
1, 2
Undefined
```

```
> [a, b] = [b, a]
(2) [2, 1]
> console.log(a, b)
2, 1
undefined
```

```
> let arrayA = [1, 2, 3, 4, 5]
undefined
> const [a, b, c] = arrayA
undefined
> console.log(a, b, c)
1 2 3
undefined
```

## SECTION 6-3 객체와 배열 고급(5)

### ◦ 객체 기반의 다중 할당

- 최신 자바스크립트에서는 객체 내부에 있는 속성을 꺼내서 변수로 할당할 때 다음과 같이 사용 가능.

```
{속성 이름, 속성 이름} = 객체  
{식별자=속성 이름, 식별자=속성 이름} = 객체
```

- 객체 속성 꺼내서 다중 할당하기 (소스 코드 6-3-3.html)

```
01 <script>  
02  // 객체를 생성합니다.  
03  const object = {  
04    name: '혼자 공부하는 파이썬',  
05    price: 18000,  
06    publisher: '한빛미디어'  
07  }  
08  
09  // 객체에서 변수를 추출합니다.  
10  const {name, price} = object  
11  console.log('# 속성 이름 그대로 꺼내서 출력하기')  
12  console.log(name, price)  
13  console.log("")  
14  
15  const {a=name, b=price} = object  
16  console.log('# 다른 이름으로 속성 꺼내서 출력하기')  
17  console.log(a, b)  
18 </script>
```

→ name 속성과 price 속성을 그대로 꺼냄

→ name 속성을 a, price 속성을 b라는 이름으로 꺼냄

실행 결과

```
# 속성 이름 그대로 꺼내서 출력하기  
혼자 공부하는 파이썬 18000  
  
# 다른 이름으로 속성 꺼내서 출력하기  
혼자 공부하는 파이썬 18000
```

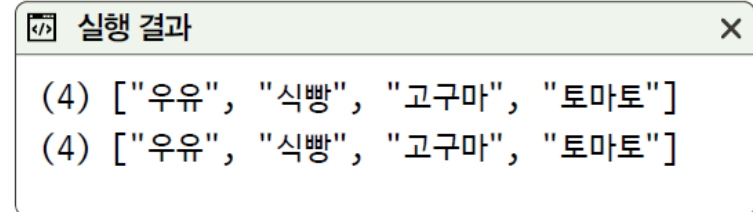
## SECTION 6-3 객체와 배열 고급(6)

### 배열 전개 연산자

- 배열과 객체는 할당할 때 **얕은 복사**라는 것이 이루어짐.
- '물건\_200301'라는 배열을 '물건\_200302'로 복사한 뒤에 '물건\_200302'에 push() 메소드를 호출해서 자료를 추가했다. 그런 다음 '물건\_200301'과 '물건\_200302'를 출력하면 어떤 값을 출력할까?
- 얕은 복사 이해하기 (소스 코드 6-3-4.html)

```
01 <script>
02  // 사야 하는 물건 목록
03  const 물건_200301 = ['우유', '식빵']
04  const 물건_200302 = 물건_200301
05  물건_200302.push('고구마')
06  물건_200302.push('토마토')
07
08  // 출력
09  console.log(물건_200301)
10  console.log(물건_200302)
11 </script>
```

- 정답은 같은 값이 나온다.
- 배열은 복사해도 다른 이름이 붙을 뿐. 이를 **얕은 복사(참조 복사)**라고 부른다.



## SECTION 6-3 객체와 배열 고급(7)

- 얇은 복사의 반대말은 **깊은 복사**.
- 깊은 복사는 복사한 두 배열이 완전히 독립적으로 작동. 자바스크립트 개발에서는 '클론(clone)을 만드는 것'이라고 표현하기도 함.
- 과거에는 깊은 복사를 위해 반복문을 활용한 긴 코드를 사용하기도 했지만, 최신 자바스크립트에서는 전개 연산자로도 가능

---

[...배열]

---

- 전개 연산자를 사용해 배열 복사하기 (소스 코드 6-3-5.html)

---

```
01 <script>
02  // 사야 하는 물건 목록
03  const 물건_200301 = ['우유', '식빵']
04  const 물건_200302 = [...물건_200301]
05  물건_200302.push('고구마')
06  물건_200302.push('토마토')
07
08  // 출력
09  console.log(물건_200301)
10  console.log(물건_200302)
11 </script>
```

---

실행 결과

(2) ["우유", "식빵"]

(4) ["우유", "식빵", "고구마", "토마토"]

## SECTION 6-3 객체와 배열 고급(8)

- 복사한 뒤에 자료를 추가하는 코드도 많이 사용되므로 전개 연산자로 배열을 전개하고 뒤에 자료를 추가하는 패턴도 사용 가능.

[...배열, 자료, 자료, 자료]

- 전개 연산자로 배열 전개하고 자료 추가하기 (소스 코드 6-3-6.html)

```
01 <script>
02  // 사야 하는 물건 목록
03  const 물건_200301 = ['우유', '식빵']
04  const 물건_200302 = ['고구마', ...물건_200301, '토마토']
05
06  // 출력
07  console.log(물건_200301)
08  console.log(물건_200302)
09 </script>
```

해당 위치에 복사되어 전개.  
위치를 원하는 곳에 놓아서 요소들의 순서를 바꿀 수 있음.

실행 결과

```
(2) ["우유", "식빵"]
(4) ["고구마", "우유", "식빵", "토마토"]
```

- 전개 연산자를 입력한 곳에 배열이 전개되어 들어가는 것이므로 배열을 여러 번 전개 가능. 다른 2개 이상의 배열을 붙일 때도 활용.

```
> const a = ['우유', '식빵']
undefined
> const b = ['고구마', '토마토']
Undefined

> [...a, ...b]
(4) ["우유", "식빵", "고구마", "토마토"]
> [...b, ...a]
(4) ["고구마", "토마토", "우유", "식빵"]
```



## SECTION 6-3 객체와 배열 고급(9)

### ◦ 객체 전개 연산자

- 객체도 깊은 복사를 할 때 전개 연산자 사용 가능

`{...객체}`

- 얇은 복사로 객체 복사하기 (소스 코드 6-3-7.html)

```
01 <script>
02   const 구름 = {
03     이름: '구름',
04     나이: 6,
05     종족: '강아지'
06   }
07   const 별 = 구름
08   별.이름 = '별'
09   별.나이 = 1
10
11   console.log(JSON.stringify(구름))
12   console.log(JSON.stringify(별))
13 </script>
```

실행 결과

```
{"이름": "별", "나이": 1, "종족": "강아지"}
{"이름": "별", "나이": 1, "종족": "강아지"}
```

## SECTION 6-3 객체와 배열 고급(10)

- 전개 연산자를 사용해서 깊은 복사를 하면 두 객체가 독립적으로 동작.
- 전개 연산자를 사용해 깊은 복사하기 (소스 코드 6-3-8.html)

```
01 <script>
02   const 구름 = {
03     이름: '구름',
04     나이: 6,
05     종족: '강아지'
06   }
07   const 별 = {...구름}
08   별.이름 = '별'
09   별.나이 = 1
10
11   console.log(JSON.stringify(구름))
12   console.log(JSON.stringify(별))
13 </script>
```

실행 결과

```
{"이름": "구름", "나이": 6, "종족": "강아지"}
{"이름": "별", "나이": 1, "종족": "강아지"}
```

## SECTION 6-3 객체와 배열 고급(11)

- 전개 연산자를 사용해 객체 요소를 추가할 수 있음

---

`{...객체, 자료, 자료, 자료}`

---

- 변경하고 싶은 속성만 추가하기(소스 코드 6-3-9.html)

---

```
01 <script>
02   const 구름 = {
03     이름: '구름',
04     나이: 6,
05     종족: '강아지'
06   }
07   const 별 = {
08     ...구름,
09     이름: '별', // 기존의 속성 덮어 쓰기
10     나이: 1, // 기존의 속성 덮어 쓰기
11     예방접종: true
12   }
13
14   console.log(JSON.stringify(구름))
15   console.log(JSON.stringify(별))
16 </script>
```

---

실행 결과

```
{"이름": "구름", "나이": 6, "종족": "강아지"}
{"이름": "별", "나이": 1, "종족": "강아지", "예방접종": true}
```

## SECTION 6-3 객체와 배열 고급(12)

- 객체는 전개 순서가 중요. '전개'라는 이름처럼 전개한 부분에 객체가 펼쳐지기 때문.
- 다음과 같이 입력한 경우 '구름'이라는 객체가 앞부분에 전개. 따라서 '뒤에 있는 이름과 나이'가 '앞에 있는 이름과 나이'를 덮어씀.

```
const 별 = {  
  ...구름,  
  이름: '별',  
  나이: 1,  
  예방접종: true  
}
```

```
const 별 = {  
  이름: '구름',  
  나이: 6,  
  종족: '강아지'  
  
  이름: '별',  
  나이: 1,  
  예방접종: true  
}
```

- 전개를 뒤에 한다면 뒤에서 전개됨. '뒤에 있는 이름과 나이'가 '앞에 있는 이름과 나이'를 덮어씀.

```
const 별 = {  
  이름: '별',  
  나이: 1,  
  예방접종: true,  
  ...구름  
}
```

```
const 별 = {  
  이름: '별',  
  나이: 1,  
  예방접종: true,  
  
  이름: '구름',  
  나이: 6,  
  종족: '강아지'  
}
```

## SECTION 6-3 객체와 배열 고급(13)

- 소스 코드 6-3-9.html의 전개 부분을 뒤로 옮기면 다음과 같음
- 전개 부분 뒤로 이동하기 (소스 코드 6-3-10.html)

```
01 <script>
02   const 구름 = {
03     이름: '구름',
04     나이: 6,
05     종족: '강아지'
06   }
07   const 별 = {
08     이름: '별',
09     나이: 1,
10     예방접종: true,
11     ...구름
12   }
13
14   console.log(JSON.stringify(구름))
15   console.log(JSON.stringify(별))
16 </script>
```

실행 결과

```
{"이름": "구름", "나이": 6, "종족": "강아지"}
{"이름": "구름", "나이": 6, "예방접종": true, "종족": "강아지"}
```

# [마무리]

## ◦ 4가지 키워드로 정리하는 핵심 포인트

- 속성 존재 여부 확인은 객체 내부에 어떤 속성이 있는지 확인하는 것을 의미. 객체에 없는 속성은 접근하면 undefined가 나오는데, 이를 활용하면 됨.
- 다중 할당은 배열과 객체 하나로 여러 변수에 값을 할당하는 것을 의미.
- 얕은 복사(참조 복사)는 복사하는 행위가 단순히 다른 이름을 붙이는 형태로 동작하는 복사를 의미.
- 깊은 복사는 복사 후 두 객체를 완전하게 독립적으로 사용할 수 있는 복사를 의미합니다.

## ◦ 확인 문제

1. 다음 중 전개 연산자의 형태로 올바른 것은?

① ~

② ...

③ @

④ spread

2. 구글에 “popular javascript libraries 2020” 등으로 검색해서 자바스크립트 라이브러리를 살펴본 후, 이름을 7개 적기(이름만 적지 말고 어떤 라이브러리인지도 꼭 살펴보기).