

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 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 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(object)

- 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', → 키와 값 뒤에 쉼표(,)를
  유형: '당절임',         넣어 구분
  성분: '망고, 설탕, 메타중아황산나트륨, 치자황색소',
  원산지: '필리핀'
}
</script>
```

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

객체 생성 방법

```
const array = [100, 20, '문자열', true, function () {}, () => {}]  
const object = {  
  키: 값,  
  키: 값,  
  키: 값,  
  키: 값  
}
```

식별자

1. 숫자로 시작하지 않는다.
2. 기호는 \$와 _만 포함한다.

```
< 6-4-10.html > ...  
1 <script>  
2  
3   const array = [100, 20, '문자열', true, function () {}, () => {}]  
4   console.log(array[0])  
5   array[0] = 200  
6   console.log(array[0])  
7  
8   const object = {  
9     name: '구름',  
10    age: 7  
11  }  
12  
13  console.log(object.name)  
14  object.name = '별'  
15  object.age = 1  
16  console.log(object.name)  
17  console.log(object.age)  
18  
19 </script>
```

100
200
구름
별
1
content loaded

SECTION 6-1 객체의 기본(2)

◦ 객체(object)

▪ 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)

◦ 속성(property)과 메소드(method)

- 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>
```

```
// 속성(property)
// 메서드(method)

const dog = {
  name: '구름',
  age: 7,
  bark: function () {
    console.log(`${dog.name}이/가 짖습니다.`)
  }
}

dog.bark()
```

구름이/가 짖습니다.

content loaded



익명 함수와 화살표 함수 이용 – bark 메서드, sleep 메서드

```
// 속성(property)
// 메서드(method)
const dog = {
  name: '구름',
  age: 7,
  bark: function () {
    console.log(`${dog.name}이/가 짖습니다.`)
  },
  sleep: () => {
    console.log(`${dog.name}이/가 잡니다.`)
  }
}

dog.bark()
dog.sleep()
```

구름이/가 짖습니다.

구름이/가 잡니다.

content loaded

>

This 사용

```
// 속성(property)
// 메서드(method)
const dog = {
  name: '구름',
  age: 7,
  bark: function () {
    // console.log(`${dog.name}이/가 짖습니다.`)
    // console.log(`${this.name}이/가 짖습니다.`)
    console.log(this)
  },
  sleep: () => {
    // console.log(`${dog.name}이/가 잡니다.`)
    console.log(this)
  }
}

dog.bark()
dog.sleep()
```

```
▼ Object 1
  age: 7
  ▶ bark: f (/
  name: "구름"
  ▶ sleep: () => {...}
  ▶ [[Prototype]]: Object
```

```
▼ Window 1
  ▶ alert: f alert(,
  ▶ atob: f atob(,
  ▶ blur: f blur(,
  ▶ btoa: f btoa(,
  ▶ caches: CacheStorage {}
  ▶ cancelAnimationFrame: f cancelAnimationFrame(,
  ▶ cancelIdleCallback: f cancelIdleCallback(,
  ▶ captureEvents: f captureEvents(,
  ▶ chrome: {loadTimes: f, csi: f}
  ▶ clearInterval: f clearInterval(,
  ▶ clearTimeout: f clearTimeout(,
  ▶ clientInformation: Navigator {vendorSub: '', productSub: '20030107', vendor: 'G
  ▶ close: f close(,
  closed: false
  ▶ confirm: f confirm(,
  ▶ cookieStore: CookieStore {onChange: null}
  ▶ createImageBitmap: f createImageBitmap(,
  credentialless: false
  crossOriginIsolated: false
  ▶ crypto: Crypto {subtle: SubtleCrypto}
  ▶ customElements: CustomElementRegistry {}
  devicePixelRatio: 1
  ▶ document: document
  ▶ documentPictureInPicture: DocumentPictureInPicture {window: null, onenter: nul
  event: undefined
  ▶ external: External {}
  fence: null
  ▶ fetch: f fetch(,
  ▶ find: f find(,
  ▶ focus: f focus(,
  frameElement: null
  ▶ frames: Window {window: Window, self: Window, document: document, name: '', locat
  ▶ getComputedStyle: f getComputedStyle(,
  ▶ getScreenDetails: f getScreenDetails(,
  ▶ getSelection: f getSelection(,
  ▶ history: History {length: 1, scrollRestoration: 'auto', state: null}
  ▶ indexedDB: IDBFactory {}
  innerHeight: 925
```

This 바인딩

```
// 익명함수 : this 바인딩을 한다.  
// 화살표 함수 : this 바인딩을 안 한다.  
  
// ## this 바인딩 : this를 현재 객체와 연결하는 행위  
  
// 객체 자신을 나타낼 때 this.을 생략하면 안 됩니다.
```

구름이/가 짊습니다.

이/가 잡니다.

▶ Window

content loaded






```
const dog = {  
  name: '구름',  
  age: 7,  
  bark: function () {  
    console.log(`${this.name}이/가 짊습니다.`)  
  },  
  sleep: () => {  
    console.log(`${this.name}이/가 잡니다.`)  
    console.log(this)  
  }  
}  
  
dog.bark()  
dog.sleep()
```

SECTION 6-1 객체의 기본(4)

- 속성(property)과 메소드(method)
 - 메소드 내부에서 **this 키워드** 사용하기
 - 자기 자신의 자신이 가진 속성이라는 것을 표시할 때 **this 키워드**를 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

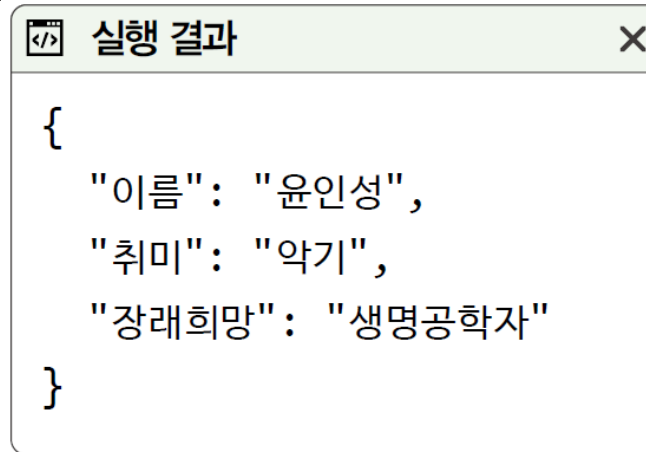
 실행 결과 

구름은/는 밥을/를 먹습니다.

SECTION 6-1 객체의 기본(5)

- 동적으로 객체 속성 추가/제거 → 객체에 동적으로 속성 추가 및 삭제
 - 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"
}
```

처음 만들 때 같이 만드는 것 -> 정적으로 생성

나중에 만드는 것 -> 동적으로 생성

동적으로 속성 추가

- 객체.속성 = 값
- 객체["속성"] = 값

동적으로 객체 속성 추가/삭제

```
// 객체의 키와 값을 정적으로 생성한다.  
const pet = {  
  name: '구름',  
  age: 8,  
}  
  
// 객체의 키와 값을 동적으로 생성한다.  
pet.color = 'brown'  
  
// 출력  
console.log(pet)
```

```
▼ {name: '구름', age: 8, color: 'brown'} ⓘ  
  age: 8  
  color: "brown"  
  name: "구름"  
  ► [[Prototype]]: Object  
content loaded
```

```
// 객체의 키와 값을 정적으로 생성한다.  
const pet = {  
  name: '구름',  
  age: 8,  
}  
  
// 객체의 키와 값을 동적으로 생성한다.  
pet.color = 'brown'  
  
// 객체의 키와 값을 동적으로 제거한다.  
delete pet.color  
  
// 출력  
console.log(pet)
```

```
▼ Object ⓘ  
  age: 8  
  name: "구름"  
  ► [[Prototype]]: Object  
content loaded
```

SECTION 6-1 객체의 기본(6)

- 동적으로 객체 속성 추가/제거

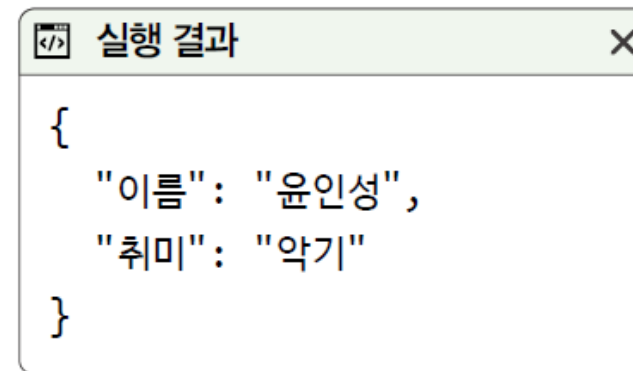
- 동적으로 객체 속성 제거하기

- 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>
```



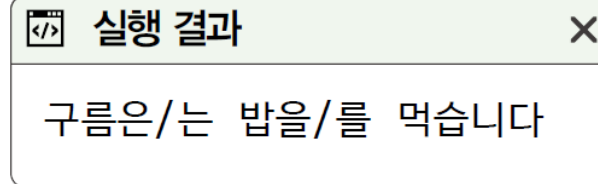
```
{
  "이름": "윤인성",
  "취미": "악기"
}
```

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>
```

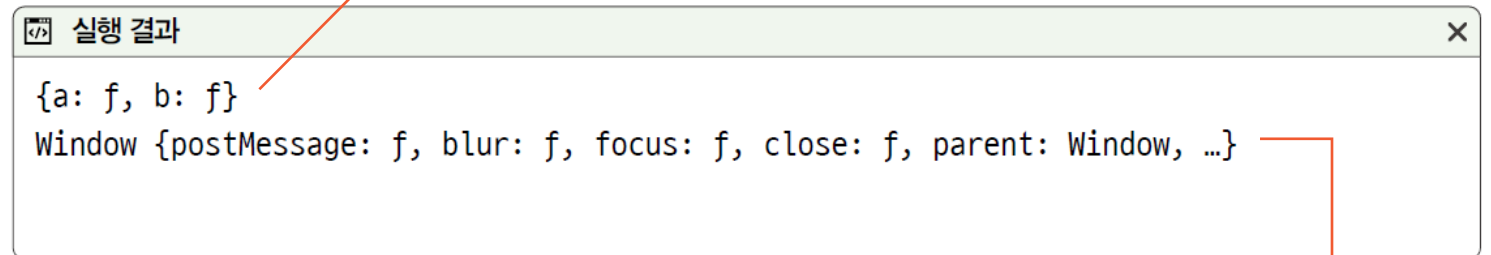


[좀 더 알아보기] 화살표 함수를 사용한 메소드

- `function () {}` 형태로 선언하는 **익명 함수**와 `() => {}` 형태로 선언하는 **화살표 함수**는 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

- 요소(element)란 배열 내부에 있는 값을 의미
- 속성(property)은 객체 내부에 있는 값을 의미
- 메소드(method)는 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

```
1  const book = {  
2    name: '혼자 공부하는 파이썬',  
3    price: 18000,  
4    publisher: '한빛미디어'  
5  }
```

```
const book = { name: '혼자 공부하는 파이썬', price: 18000, publisher: '한빛미디어' }
```

2. 다음 중 객체에 동적으로 속성을 추가하는 문법은? **객체.속성**

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?

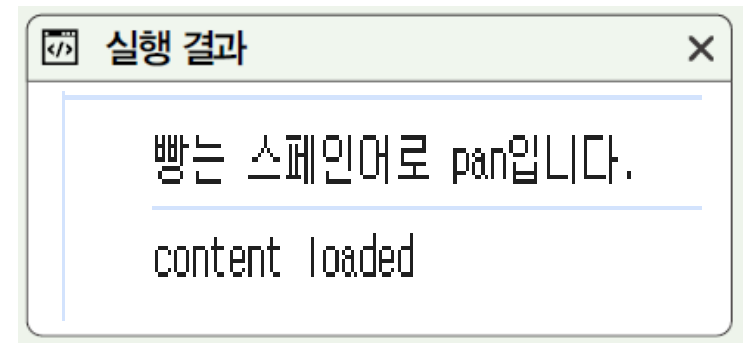
① **delete 객체[속성]** ② 객체.delete('속성') ③ delete 객체 from 속성 ④ delete 속성 from 객체

[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.`)
  }    -> 메소드 부분
}
object.print('es')
</script>
```



4번 코딩문

<> 6-4-8.html > ...

```
1  <script>
2      const object = {
3          ko: '빵',
4          en: 'bread',
5          ja: 'パン',
6          fr: 'pain',
7          es: 'pan',
8          lang: {
9              ko: '한국어',
10             en: '영어',
11             ja: '일본어',
12             fr: '프랑스어',
13             es: '스페인어'
14         },
15         print: function (lang) {
16             console.log(`${this.ko}는 ${this.lang[lang]}로 ${this[lang]}입니다.`)
17         }
18     }
19     object.print('es')
20 </script>
```

4번 풀이

```
1 const object = {
2   ko: '빵',
3   en: 'bread',
4   ja: 'パン',
5   fr: 'pain',
6   es: 'pan',
7   lang: {
8     ko: '한국어',
9     en: '영어',
10    ja: '일본어',
11    fr: '프랑스어',
12    es: '스페인어'
13  },
14  print: function (lang = 'es') {
15    console.log(`${this.ko = '빵'}는 ${this.lang[lang]}로 ${this[lang]}입니다.`)
16  }
17 }
18
19 object.print('es')
```

Annotations:

- `this.ko` points to `ko: '빵'` (line 2)
- `this.lang['es']` points to `es: '스페인어'` (line 12)

```
1 const object = {
2   ko: '빵',
3   en: 'bread',
4   ja: 'パン',
5   fr: 'pain',
6   es: 'pan',
7   lang: {
8     ko: '한국어',
9     en: '영어',
10    ja: '일본어',
11    fr: '프랑스어',
12    es: '스페인어'
13  },
14  print: function (lang = 'es') {
15    console.log(`${this.ko = '빵'}는 ${this.lang[lang]}로 ${this[lang]}입니다.`)
16  }
17 }
18
19 object.print('es')
```

Annotations:

- `this.ko` points to `ko: '빵'` (line 2)
- `this.lang['es']` points to `es: '스페인어'` (line 12)

```
print: function (lang = 'es') {
  console.log(`${this.ko = '빵'}는 ${this.lang[lang] = '스페인어'}로 ${this[lang] = 'pan'}입니다.`)
}
```

SECTION 6-2 객체의 속성과 메소드 사용하기(1)

◦ 객체 자료형 (숫자, 문자열, 불 빼고는 모두 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

기본 자료형

- 숫자
- 문자열
- 불

객체 자료형

- 함수
- 배열
- 객체

```
> const b = () => {}
```

```
< undefined
```

```
> typeof(b)
```

```
< 'function'
```

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 -> 다음 페이지에 자세한 예제가 있음

```
> 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

기본 자료형과 객체 자료형의 차이

기본 자료형 → 스택에 값을 저장

- 숫자
- 문자열
- 불

객체 자료형 → 스택과 힙을 연결 → 속성과 메서드

- 함수
- 배열
- 객체
- 등등

```
> const a = [ ]
```

```
< undefined
```

```
> a.내맘대로속성 = 10
```

```
< 10
```

```
> a.내맘대로속성
```

```
< 10
```

```
> const a = [ ]
```

```
< undefined
```

```
> a.내맘대로속성 = 10
```

```
< 10
```

```
> a.내맘대로속성
```

```
< 10
```

```
> delete a.내맘대로속성
```

```
< true
```

```
> a.내맘대로속성
```

```
< undefined
```

```
> const b = function () { }
```

```
< undefined
```

```
> b.value = 10
```

```
< 10
```

```
> b.value
```

```
< 10
```

```
> const c = 10
```

```
< undefined
```

```
> c.value = 20
```

```
< 20
```

```
> c.value
```

```
< undefined
```

기본자료형을 객체로 선언하기

- `const 객체 = new 객체 자료형 이름()`

```
> const a = new Number(10)
```

```
< undefined
```

```
> const b = new String('문자열')
```

```
< undefined
```

```
> const c = new Boolean(true)
```

```
< undefined
```

```
> typeof(a) + ' ' + typeof(b) + ' ' + typeof(c)
```

```
< 'object object object'
```

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

> const a = []

< undefined

> typeof (a)

< 'object'

> Array.isArray(a)

< true

> Array.isArray(10)

< false

> Array.isArray({})

< false

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

```
> const a = '안녕하세요'
```

```
< undefined
```

```
> a.length
```

```
< 5
```

```
> a.bold()
```

```
< '<b>안녕하세요</b>'
```

```
> a.anchor('최일준')
```

```
< '<a name="최일준">안녕하세요</a>'
```

속성과 메서드가 좋은 기능이므로

기본 자료형 뒤에 점을 찍고 속성과 메서드를 쓰면

-일시적으로- 객체로 변경해서 속성과 메서드를 쓸

수 있게 해줍니다.

이론적으로는 기본 자료형은 속성과 메서드를 추가할 수 없고,

기본 자료형이 속성과 메서드를 사용할 수 있는 이유는

일시적으로 객체 자료형으로 변환되어서 사용할 수 있는 건데,

이건 몰라도 되는데 지금까지 점 찍고 array.length를 사용하는데

아무런 문제가 없으니, 그냥 그러려니 하면 됨.

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.  
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>
```

```
1 <script>  
2 String.prototype.value = 10  
3  
4 const a = '문자열'  
5 console.log(a.value)  
6 </script>
```

10

content loaded

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
```

프로토타입으로 문자열 메소드 추가하기, 다른 예제

```
String.prototype.contain = function(다른문자열) {  
  | return this.indexOf(다른문자열) >= 0  
  |  
}  
  
const a = '문자열'  
console.log(`a.contain('문자'): ${a.contain('문자')}`)
```

a.contain('문자'): true

content loaded



한번만 프로토타입에 이렇게 정의해주면

“문자열.contain(다른문자열)”으로 간단히 쓸 수 있고

코드의 의미도 명확해지므로 사용하기 좋음

-> 많이 사용하지 않으나, 사용하기도 함.

String.prototype으로 문자열 메서드를 추가하는 이유

- 기본으로 제공하는 데이터 타입에 사용자 정의 메서드를 추가해 재사용성과 가독성을 높이기 위해서
- 주요 이유와 장점
- **재사용성 향상**
- 프로토타입에 메서드를 추가하면 해당 메서드가 **모든 문자열 객체에서 사용 가능**하게 됨. 예) String.prototype.contains를 추가하면, 문자열을 다루는 다양한 코드에서 일관된 방식으로 contains 메서드를 사용할 수 있음. 이로 인해 여러 곳에서 동일한 기능을 필요로 할 때 매번 같은 코드를 반복할 필요가 없어짐.
- **가독성 향상 및 코드 간결화**
- 특정 기능을 수행하는 메서드를 만들고, 이를 프로토타입에 추가하면 코드가 보다 **읽기 쉬워지고 이해하기 쉬워짐**. 예) contains이라는 메서드가 있다면, 문자열에 해당 부분 문자열이 포함되는지를 직관적으로 표현할 수 있음.
- **유틸리티 메서드 추가**
- JavaScript 기본 String 메서드에 없는 유틸리티 메서드를 직접 추가할 수 있음. 예) contains 외에도 문자열 변형, 검색 기능 등의 커스텀 메서드를 만들고자 할 때 유용함. 내장 메서드에서 제공되지 않는 기능을 사용자 정의 방식으로 사용할 수 있음.
- **코드 일관성 확보**
- 여러 모듈이나 파일에서 문자열과 관련된 동일한 기능을 일관되게 사용할 수 있음. 즉, 모든 문자열 객체에서 동일한 메서드를 호출할 수 있으므로, 코드가 일관되며 유지 보수도 쉬워짐.

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

```
> const a = 123.456789
```

```
< undefined
```

```
> a.toFixed(3)
```

```
< '123.457'
```

```
> a.toFixed(2)
```

```
< '123.46'
```

```
> a.toFixed(1)
```

```
< '123.5'
```

```
> a.toFixed()
```

```
< '123'
```

“개발자는 구글 없으면 개발 못한다.” 라는 말
-> 검색하면 모든 것이 다 나온다.
-> 요즘 생성형 AI를 활용해도 자세히 알려준다.

SECTION 6-2 객체의 속성과 메소드 사용하기(9)

◦ Number 객체

- NaN과 Infinity 확인하기: `isNaN()`, `isFinite()`
 - Number 뒤에 점(.)을 찍고 사용

`isNaN()`

```
> m
NaN
> m === NaN
false
> Number.isNaN(m)
true
```

NaN을 생성

NaN과 비교해서는
NaN인지 확인 불가

```
> Number.isNaN()
< false
> Number('안녕하세요')
< NaN
> const a = Number('안녕하세요')
< undefined
> Number.isNaN(a)
< true
> Number.isNaN(20)
< false
```

`is○○()` 메서드는 true 또는 false를 리턴함

`isFinite()`

```
> const n = 10 / 0
undefined
> n
Infinity
> const o = -10 / 0
undefined
> o
-Infinity
```

양의 무한대를 생성

음의 무한대를 생성

```
> Number.isFinite(n)
false
> Number.isFinite(o)
false
> Number.isFinite(1)
true
> Number.isFinite(10)
true
```

NaN(Not a Number)

-> 자료형은 숫자인데
값이 숫자가 아닌 것

isFinite(유한한 숫자인가?)가 false로 나옴

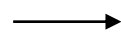
일반적인 숫자는 셀 수 있으므로
true가 나옴

SECTION 6-2 객체의 속성과 메소드 사용하기(10)

◦ String 객체

- 문자열 양쪽 끝의 공백 없애기: `trim()`

“ 안녕하세요 ”



“안녕하세요”

```
> 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

```
> `
  OHO
.trim()
< 'OHO'
> `   OHO   `.trim()
< 'OHO'`
```

[참조] 모질라 String 객체의 속성과 메소드

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

Split() 예시

```
> //split()
```

```
← undefined
```

```
> '21.05.02,윤민성,김밥집,2000'
```

```
← '21.05.02,윤민성,김밥집,2000'
```

```
> '21.05.02,윤민성,김밥집,2000'.split(',')
```

```
← ▼ (4) ['21.05.02', '윤민성', '김밥집', '2000'] ⓘ
```

```
0: "21.05.02"
```

```
1: "윤민성"
```

```
2: "김밥집"
```

```
3: "2000"
```

```
length: 4
```

```
▶ [[Prototype]]: Array(0)
```

```
> '21.05.02 윤민성 김밥집 2000'.split(' ')
```

```
← ▼ (4) ['21.05.02', '윤민성', '김밥집', '2000'] ⓘ
```

```
0: "21.05.02"
```

```
1: "윤민성"
```

```
2: "김밥집"
```

```
3: "2000"
```

```
length: 4
```

```
▶ [[Prototype]]: Array(0)
```

```
> '21.05.02|윤민성|김밥집|2000'.split('|')
```

```
← ▼ (4) ['21.05.02', '윤민성', '김밥집', '2000'] ⓘ
```

```
0: "21.05.02"
```

```
1: "윤민성"
```

```
2: "김밥집"
```

```
3: "2000"
```

```
length: 4
```

```
▶ [[Prototype]]: Array(0)
```

trim() , split() 예시

```
> const input = `
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
`

< undefined

> input.trim()

< '일자,달러,엔,유로\n02,1141,8,1097,46,1262,37\n03,1148,7,1111,36,1274,65\n04,1140,6,1107,81,1266,58\n07,1143,4,1099,58,1267,8\n08,1141,6,1091,97,1261,07'

> input.trim().split('\n')

< (6) ['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']
  0: "일자,달러,엔,유로"
  1: "02,1141,8,1097,46,1262,37"
  2: "03,1148,7,1111,36,1274,65"
  3: "04,1140,6,1107,81,1266,58"
  4: "07,1143,4,1099,58,1267,8"
  5: "08,1141,6,1091,97,1261,07"
  length: 6
  ▶ [[Prototype]]: Array(0)
```

SECTION 6-2 객체의 속성과 메소드 사용하기(12)

◦ JSON 객체

- 인터넷에서 문자열로 데이터를 주고 받을 때는 CSV(Comma-Separated Values), XML(Extensible Markup Language), CSON(CoffeeScript Object Notation) 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
 - 여러 개의 자료

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

```
[{  
  "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()** 메소드(스트링어파이, 스트리니파이)를 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>
```

```
const obj = {name: "홍길동", age: 25};
const jsonstring = JSON.stringify(obj);
console.log(jsonstring);
```

```
{"name": "홍길동", "age": 25}
content loaded
```

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

들여쓰기 2칸으로 설정

```
JSON.stringify(value, replacer, space)
```

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

들여쓰기 2칸이 추가

JSON.stringify() 메서드의 기본 문법

기본 문법

javascript

 코드 복사

```
JSON.stringify(value, replacer, space)
```

- **value**: JSON 문자열로 변환할 JavaScript 값 (객체, 배열 등)
- **replacer** (선택): 문자열 변환 과정에서 특정 속성만 포함하거나 변경할 수 있도록 하는 함수 또는 배열
- **space** (선택): JSON 문자열의 들여쓰기 공백 수. 숫자를 지정하면 해당 수만큼 들여쓰기, 문자열을 지정하면 그 문자열을 들여쓰기에 사용합니다.

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)
```

JSON.parse() 메소드를 사용법

기본 문법

javascript

 코드 복사


```
JSON.parse(text, reviver)
```

- **text**: JSON 형식의 문자열. 반드시 유효한 JSON 형식이어야 합니다.
- **reviver** (선택): 결과 객체를 변환할 수 있는 함수로, 각 값이 객체에 추가되기 전에 호출됩니다.

사용 예시

기본 사용법

javascript

 코드 복사

```
const jsonString = '{"name": "Alice", "age": 25, "city": "Seoul"}';
const obj = JSON.parse(jsonString);

console.log(obj);
// 출력: { name: 'Alice', age: 25, city: 'Seoul' }
console.log(obj.name); // 출력: Alice
```

SECTION 6-2 객체의 속성과 메소드 사용하기(16)

◦ Math 객체

- 수학과 관련된 기본적인 연산을 할 때는 **Math 객체를 사용**
 - Math 객체 속성으로는 **pi, e**와 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

Math 객체를 이용한 예제들

```
> Math.E
< 2.718281828459045
> Math.PI
< 3.141592653589793
> Math.floor(10.1)
< 10
> Math.ceil(10.1)
< 11
> Math.round(10.1)
< 10
> Math.max(52, 273, 103)
< 273
> Math.min(...[52, 273, 103])
< 52
> Math.min(52, 273, 103)
< 52
> Math.max(...[52, 273, 103])
< 273
```

```
> Math.random()
< 0.7255002724505881
> Math.random()
< 0.6655027399869446
> Math.random()
< 0.019850741578864017
> Math.random()
< 0.9947392129943047
> Math.random()
< 0.8775375015502174
> Math.random()
< 0.02572430324772479
```

```
> //0-50 사이의 랜덤한 정수
< undefined
> Math.random() * 50
< 0.28458180084982576
> Math.floor(Math.random() * 50)
< 14
> Math.floor(Math.random() * 50)
< 11
> Math.floor(Math.random() * 50)
< 35
> Math.floor(Math.random() * 50)
< 16
> Math.floor(Math.random() * 50)
< 5
> Math.floor(Math.random() * 50)
< 2
> Math.floor(Math.random() * 50)
< 42
```

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

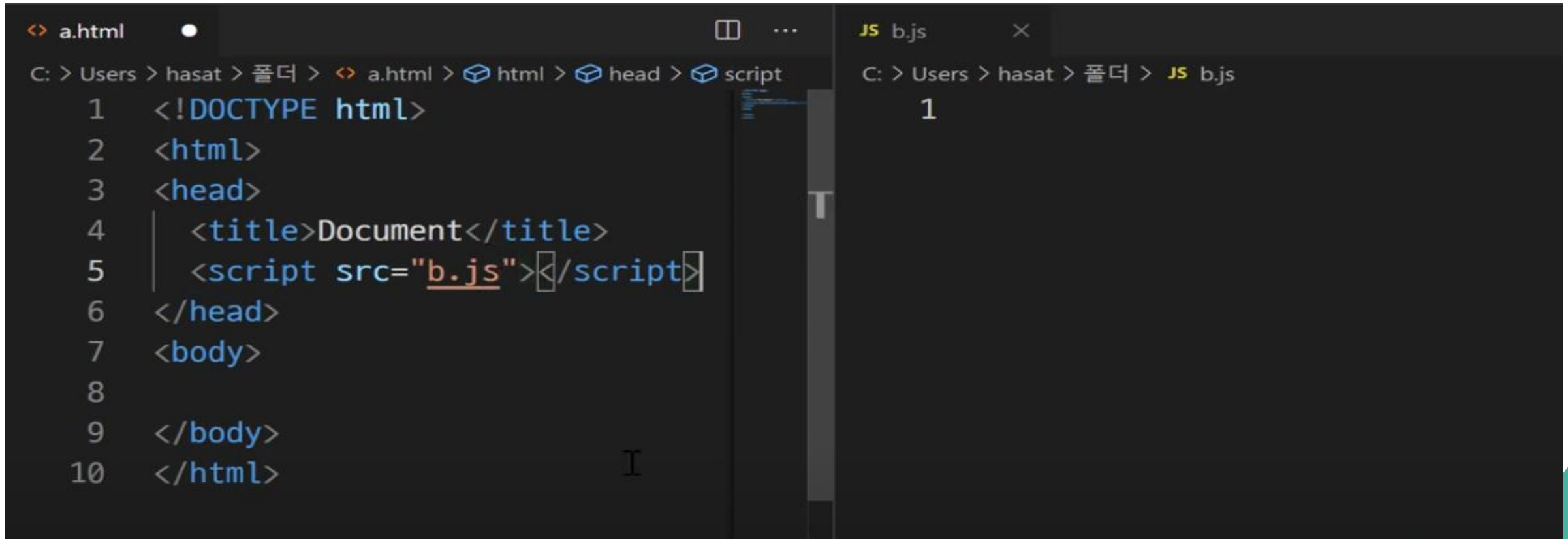
# 랜덤한 숫자 범위 확대
0~10 사이의 랜덤한 숫자: 6.780090022598715
0~50 사이의 랜덤한 숫자: 33.900450112993575

# 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



```
<> a.html
C: > Users > hasat > 폴더 > <> a.html > html > head > script
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Document</title>
5   <script src="b.js"></script>
6 </head>
7 <body>
8
9 </body>
10 </html>

JS b.js
C: > Users > hasat > 폴더 > JS b.js
1
```

코드 실행

```
a.html
C: > Users > hasat > 폴더 > a.html > html > head > script
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Document</title>
5   <script src="b.js"></script>
6 </head>
7 <body>
8
9 </body>
10 </html>

JS b.js
C: > Users > hasat > 폴더 > JS b.js
1 alert('ALERT!!')
```

Browser window: C:/Users/hasat/폴더/a.html

이 페이지 내용:
ALERT!!

I
확인

<meta charset="utf-8">

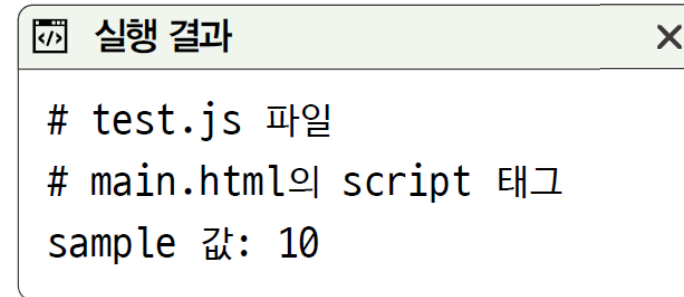
SECTION 6-2 객체의 속성과 메소드 사용하기(20)

- 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
```



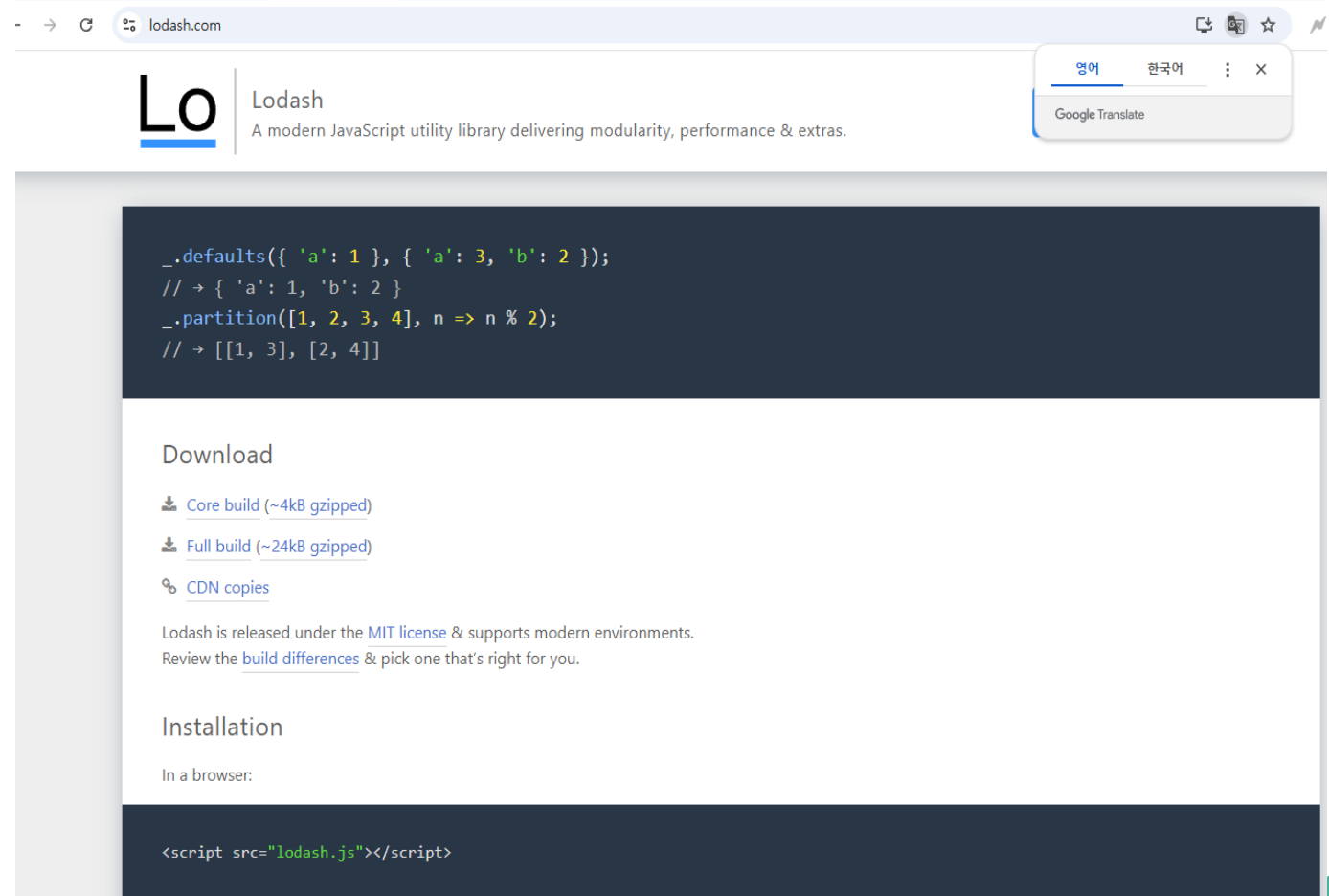
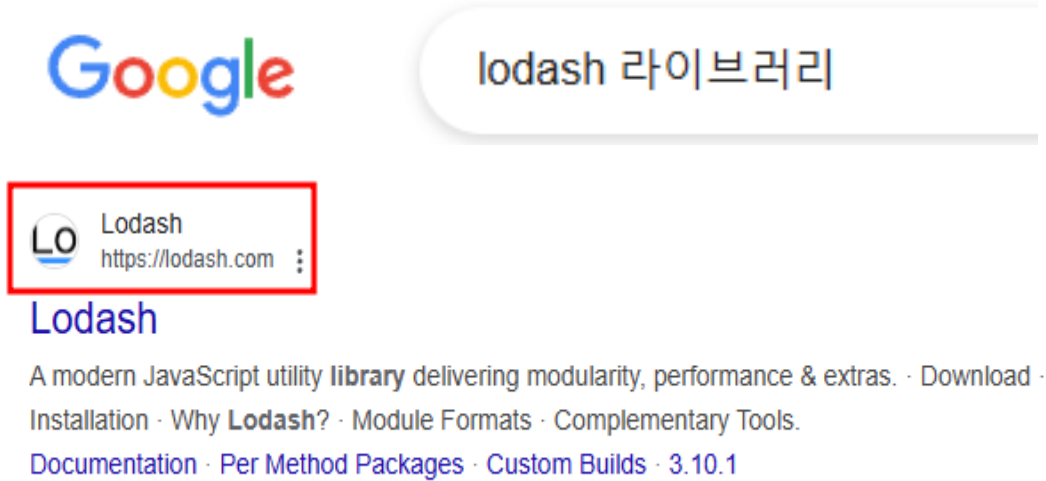
main.html 파일에서 5행의 외부 자바스크립트를 읽어들이는 script 태그(<script src="test.js"></script>)가 6~9행의 코드가 적혀 있는 script 태그보다 위에 있으므로 먼저 실행

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

Lodash 라이브러리 홈페이지 참조



CDN 카피

Download

📄 [Core build](#) (~4kB gzipped)

📄 [Full build](#) (~24kB gzipped)

🔗 [CDN copies](#)

Lodash is released under the [MIT license](#) & supports modern environments.
Review the [build differences](#) & pick one that's right for you.

INSTALL

Type: ESMDefault

```
<script src="https://cdn.jsdelivr.net/npm/lodash@4.17.21/lodash.min.js"></script>
```

Open in jsfiddleLearn more

새 탭에서 링크 열기
새 창에서 링크 열기
시크릿 창에서 링크 열기

다른 이름으로 링크 저장...
링크 주소 복사

⚡ Lightning Autofill >

Google에서 이미지 설명 가져오기 >

검사

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 }, {
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>
```

중간 생략

```
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]

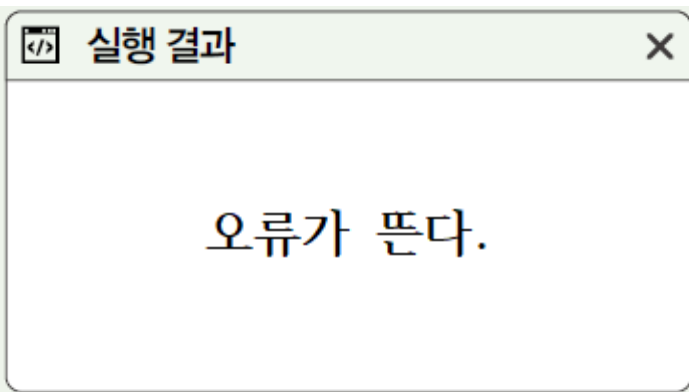
◦ 4가지 키워드로 정리하는 핵심 포인트

- 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>
```



1번 해설

```
const num = 52000;  
num.원 = function() {  
  | return this.valueOf() + '원';  
};  
console.log(num.원());
```

❌ Uncaught TypeError: num.원 is not a function
at 6-4-7.html:7:19

content loaded

52000원

content loaded



```
const num = new Number(52000)  
num.원 = function(){  
  | return this.valueOf() + '원'  
}  
console.log(num.원());
```

```
const num = 52000;  
Number.prototype.원 = function() {  
  | return this.valueOf() + '원';  
};  
console.log(num.원());
```

[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>
```

실행 결과

```
printLang("ko"): 한국어
printLang("en"): 영어
content loaded
```

```
1 <script>
2   function printLang(code) {
3     | return printLang._lang[code]
4   }
5   printLang._lang = {
6     ko: '한국어',
7     en: '영어',
8     ja: '일본어',
9     fr: '프랑스어',
10    es: '스페인어'
11  }
12  console.log('printLang("ko"):', printLang('ko'))
13  console.log('printLang("en"):', printLang('en'))
14 </script>
```

[마무리③]

◦ 확인 문제

3. 모질라 문서에서 Math 객체와 관련된 내용을 읽고 사인 90도의 값을 구하기. 참고로 **사인 90도는 1**. 아주 단순하게 생각해 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>
```

라디언값 = 각도 x Math.PI / 180

```
1  <script>  
2  const degree = 90  
3  const rad = degree * Math.PI / 180  
4  console.log(Math.sin(rad))  
5  </script>
```

1

content loaded

>

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>
```

▼ Array(4) ⓘ

- ▶ 0: {name: 'HTML5 웹 프로그래밍 입문', price: 26000, publisher: '한빛아카데미'}
- ▶ 1: {name: '딥러닝을 위한 수학', price: 25000, publisher: '위키북스'}
- ▶ 2: {name: '머신러닝 딥러닝 실전 개발 입문', price: 30000, publisher: '위키북스'}
- ▶ 3: {name: '혼자 공부하는 파이썬', price: 18000, publisher: '한빛미디어'}

length: 4

▶ [[Prototype]]: Array(0)

Live reload enabled.

content loaded

6-4-3.html:18

6-4-3.html:48

index.js:1

```
const output = _.orderBy(books, ['name'], ['asc'])
console.log(output)
```

5번 풀이

```
1 <!DOCTYPE html>
2 <html lang="ko">
3 <head>
4   <meta charset="UTF-8">
5   <title>도서 목록 정렬</title>
6   <script src="https://cdn.jsdelivr.net/npm/lodash@4.17.21/lodash.min.js"></script>
7 </head>
8 <body>
9   <script>
10    const books = [
11      { name: '혼자 공부하는 파이썬', price: 18000, publisher: '한빛미디어' },
12      { name: 'HTML5 웹 프로그래밍 입문', price: 26000, publisher: '한빛아카데미' },
13      { name: '머신러닝 딥러닝 실전 개발 입문', price: 30000, publisher: '위키북스' },
14      { name: '딥러닝을 위한 수학', price: 25000, publisher: '위키북스' }
15    ];
16
17    const output = _.orderBy(books, ['name'], ['asc']);
18    console.log(output);
19  </script>
20 </body>
21 </html>
```

오늘도 고생하셨습니다.