

- 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 심표(.)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 심표(.)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

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

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>
```

◦ 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

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>
```

실행 결과

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

SECTION 6-1 객체의 기본(6)

- 동적으로 객체 속성 추가/제거

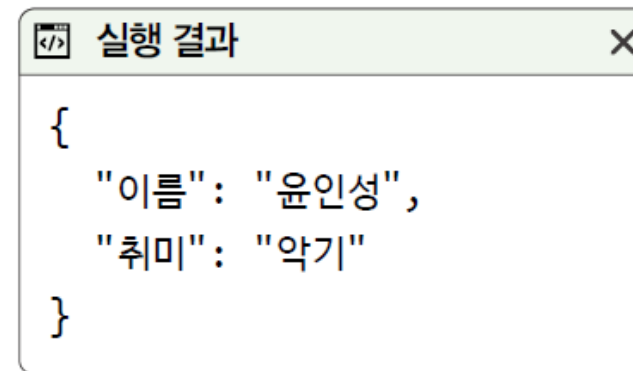
- 동적으로 객체 속성 제거하기

- 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>
```



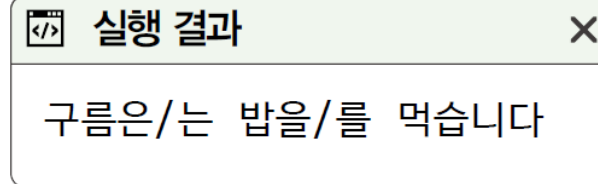
```
{
  "이름": "윤인성",
  "취미": "악기"
}
```

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>
```

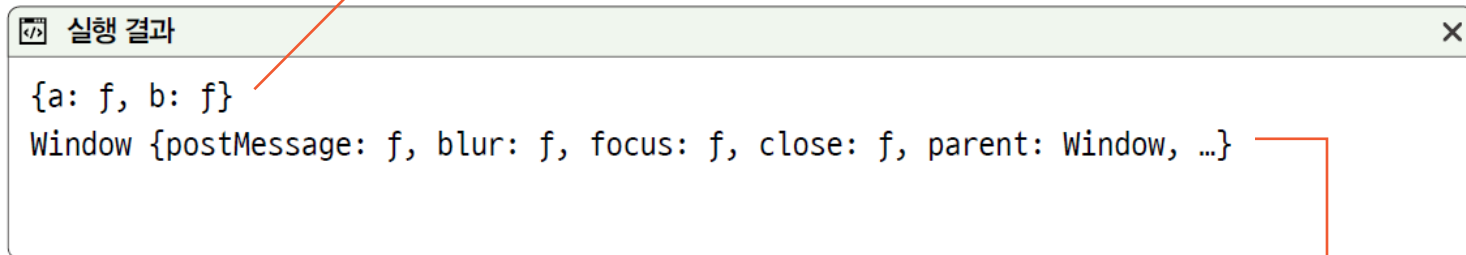


[좀 더 알아보기] 화살표 함수를 사용한 메소드

- 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

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

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<title></title>
```

```
<script>
```

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

```
console.log(typeof f)
```

```
f.sample = 10
```

```
console.log(f.sample)
```

```
console.log(f)
```

```
console.log(f + 0)
```

```
console.log(f.valueOf())
```

```
</script>
```

```
</head>
```

```
<body></body>
```

```
</html>
```

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

```
<!DOCTYPE html>
<html>
  <head>
    <title></title>
    <script>
      const h = '안녕하세요'
      h.sample = 10
      console.log( h.sample = 10)
      console.log(h.sample)
    </script>
  </head>
  <body></body>
</html>
```

```
10
```

```
undefined
```

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

```
<!DOCTYPE html>
<html>
  <head>
    <title></title>
    <script>
      const j = '안녕하세요'
      console.log(j.indexOf('하세'))
      console.log(j.indexOf('안녕'))
      console.log(j.indexOf('없는 문자열'))
    </script>
  </head>
  <body></body>
</html>
```


SECTION 6-2 객체의 속성과 메소드 사용하기(7)

프로토타입으로 메소드 추가하기



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

```
<!DOCTYPE html>
<html>
  <head>
    <title></title>
    <script>
      // contain() 메소드를 추가합니다.
      String.prototype.contain = function (data) {
        return this.indexOf(data) >= 0
      }

      Array.prototype.contain = function (data) {
        return this.indexOf(data) >= 0
      }

      // String 객체의 contain() 메소드를 사용합니다.
      const a = '안녕하세요'
      console.log('안녕 in 안녕하세요:', a.contain('안녕'))
      console.log('없는데 in 안녕하세요:', a.contain('없는데'))

      // Array 객체의 contain() 메소드를 사용합니다.
      const b = [273, 32, 103, 57, 52]
      console.log('273 in [273, 32, 103, 57, 52]:', b.contain(273))
      console.log('0 in [273, 32, 103, 57, 52]:', b.contain(0))
    </script>
  </head>
  <body></body>
</html>
```

 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

```
<!DOCTYPE html>
<html>
  <head>
    <title></title>
    <script>
      const l = 123.456789
      console.log(l.toFixed(2))
      console.log(l.toFixed(3))
      console.log(l.toFixed(4))
    </script>
  </head>
  <body></body>
</html>
```

- 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
false
> Number.isNaN(m)
true
```

NaN을 생성

NaN과 비교해서는 NaN인지 확인 불가

isFinite(유한한 숫자인가?)가 false로 나옴

일반적인 숫자는 셀 수 있으므로 true가 나옴

isFinite()

```
> const n = 10 / 0
undefined
> n
Infinity
> const o = -10 / 0
undefined
> o
-Infinity
> Number.isFinite(n)
false
> Number.isFinite(o)
false
> Number.isFinite(1)
true
> Number.isFinite(10)
true
```

양의 무한대를 생성

음의 무한대를 생성

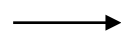
```
<!DOCTYPE html>
<html>
  <head>
    <title></title>
    <script>
      const n = 10 / 0
      console.log(n)
      const o = -10 / 0
      console.log(o)
      console.log(Number.isFinite(n))
      console.log(Number.isFinite(o))
      console.log(Number.isFinite(1))
      console.log(Number.isFinite(10))
    </script>
  </head>
  <body></body>
</html>
```

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

```
<!DOCTYPE html>
<html>
  <head>
    <title></title>
    <script>
      const stringA = `
메시지를 입력하다보니 앞에 줄바꿈도 들어가고`
      const stringB = ` 앞과 뒤에 공백도 들어가고 `
      console.log(stringA)
      console.log(stringB)
      console.log(stringA.trim())
      console.log(stringB.trim())
    </script>
  </head>
  <body></body>
</html>
```

[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

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)

실행 결과

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

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

들여쓰기 2칸이 추가

- 매개변수를 하나만 넣으면 한 줄로 변환됨
- 일반적으로 이렇게 사용

```
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>
```

>> 혼자 공부하는 자바스크립트

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 프로그  
래밍 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

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>
```

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
```

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

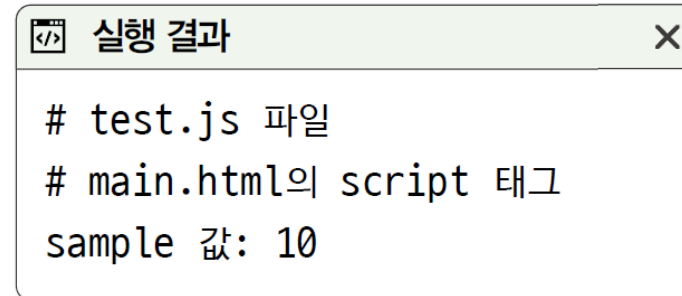
SECTION 6-2 객체의 속성과 메소드 사용하기(20)

- 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"
  }
]
```


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',
```

14

```
07   price: 18000,
```

15

```
08   publisher: '한빛미디어'
```

16

```
09 }, {
```

17

```
10   name: 'HTML5 웹 프로그래밍 입문',
```

18

```
11   price: 26000,
```

19

```
12   publisher: '한빛아카데미'
```

20

```
13 }, {
```

21

중간 생략

```
21 }]
```

22

```
22
```

23

```
23 // 가격으로 정렬한 뒤 출력합니다.
```

24

```
24 const output = _.sortBy(books, (book) => book.price)
```

25

```
25 console.log(JSON.stringify(output, null, 2))
```

26

```
26 </script>
```

```
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'
```

```
}]
```

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>
```

[마무리②]

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title></title>
5     <script>
6       // 객체를 생성합니다.
7       const object = {
8         name: '혼자 공부하는 파이썬',
9         price: 18000,
10        publisher: '한빛미디어'
11      }
12
13      // 객체 내부에 속성이 있는지 확인합니다.
14      if (object.name !== undefined) {
15        console.log('name 속성이 있습니다.')
16      } else {
17        console.log('name 속성이 없습니다.')
18      }
19      if (object.author !== undefined) {
20        console.log('author 속성이 있습니다.')
21      } else {
22        console.log('author 속성이 없습니다.')
23      }
24    </script>
25  </head>
26  <body></body>
27 </html>
28
```

[마무리②]

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title></title>
5      <script>
6        // 객체를 생성합니다.
7        const object = {
8          name: '혼자 공부하는 파이썬',
9          price: 18000,
10         publisher: '한빛미디어'
11       }
12
13       // 객체의 기본 속성을 지정합니다.
14       object.name = object.name !== undefined ? object.name : '제목 미정'
15       object.author = object.author !== undefined ? object.author : '저자 미상'
16
17       // 객체를 출력합니다.
18       console.log(JSON.stringify(object, null, 2))
19     </script>
20   </head>
21   <body></body>
22 </html>
??
```

[마무리②]

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title></title>
5     <script>
6       // 객체를 생성합니다.
7       const object = {
8         name: '혼자 공부하는 파이썬',
9         price: 18000,
10        publisher: '한빛미디어'
11      }
12
13      // 객체에서 변수를 추출합니다.
14      const { name, price } = object
15      console.log('# 속성 이름 그대로 꺼내서 출력하기')
16      console.log(name, price)
17      console.log('')
18
19      const { a=name, b=price } = object
20      console.log('# 다른 이름으로 속성 꺼내서 출력하기')
21      console.log(a, b)
22    </script>
23  </head>
24  <body></body>
25 </html>
```

[마무리②]

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title></title>
5      <script>
6        // 사야 하는 물건 목록
7        const 물건_200301 = ['우유', '식빵']
8        const 물건_200302 = 물건_200301
9        물건_200302.push('고구마')
10       물건_200302.push('토마토')
11
12       // 출력
13       console.log(물건_200301)
14       console.log(물건_200302)
15     </script>
16   </head>
17   <body></body>
18 </html>
```

[마무리②]

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title></title>
5     <script>
6       // 사야 하는 물건 목록
7       const 물건_200301 = ['우유', '식빵']
8       const 물건_200302 = [...물건_200301]
9       물건_200302.push('고구마')
10      물건_200302.push('토마토')
11
12      // 출력
13      console.log(물건_200301)
14      console.log(물건_200302)
15    </script>
16  </head>
17  <body></body>
18 </html>
19
```


[마무리②]

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title></title>
5      <script>
6        // 사야 하는 물건 목록
7        const 물건_200301 = ['우유', '식빵']
8        const 물건_200302 = ['고구마', ...물건_200301, '토마토']
9
10       // 출력
11       console.log(물건_200301)
12       console.log(물건_200302)
13     </script>
14   </head>
15   <body></body>
16 </html>
17
```

[마무리②]

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title></title>
5     <script>
6       const 구름 = {
7         이름: '구름',
8         나이: 6,
9         종족: '강아지'
10      }
11      const 별 = 구름
12      별.이름 = '별'
13      별.나이 = 1
14
15      console.log(JSON.stringify(구름))
16      console.log(JSON.stringify(별))
17    </script>
18  </head>
19  <body></body>
20 </html>
21
```