

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 6-3 객체와 배열 고급



CHAPTER 06 객체

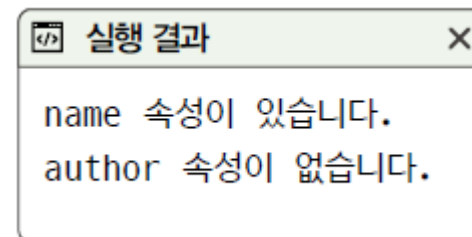
객체의 속성과 메소드, 생성, 관리하는 기본 문법 학습

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 **undefined**인지 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>
```



객체 기본값을 지정하는 경우

```
JS // 객체 기본값을 지정하는 내용! Untitled-1 ●
1  // 객체 기본값을 지정하는 내용!
2  const test = function (name, age, color) {
3    return `${name} : ${age} : ${color}`
4  }
5
6  console.log(test('구름', 7, '갈색'))
```

```
> const test = function (name, age, color) {
  return `${name} : ${age} : ${color}`
}

console.log(test('구름', 7, '갈색'))

구름 : 7 : 갈색
< undefined
```

```
const test = function (object) {
  return `${object.name} : ${object.age} : ${object.color}`
}

console.log(test({
  name: '구름',
  age: 7,
  color: '갈색'
}))
```

```
> const test = function (object) {
  return `${object.name} : ${object.age} : ${object.color}`
}

console.log(test({
  name: '구름',
  age: 7,
  color: '갈색'
}))

구름 : 7 : 갈색
< undefined
```

현대적인 개발 관점은 "개발은 코드를 입력하는 시간보다 보는 시간이 10배 이상 많으므로, 보는 것이 더 쉬워야 한다"라서 코드가 길어져도 읽기 쉬운 것을 선호합니다.

객체 기본값을 지정하는 경우

```
JS // 객체 기본값을 지정하는 내용! Untitled-1 ●
1  // 객체 기본값을 지정하는 내용!
2  const test = function (name, age, color, status = '이상 없음') {
3    return `${name} : ${age} : ${color} : ${status}`
4  }
5  console.log(test('구름', 7, '갈색'))
6
7  const test = function (object) {
8    return `${object.name} : ${object.age} : ${object.color} : ${object.status}`
9  }
10 console.log(test({
11   name: '구름',
12   age: 7,
13   color: '갈색'
14 }))
```

```
> const test = function (name, age, color, status = '이상 없음') {
    return `${name} : ${age} : ${color} : ${status}`
  }
  console.log(test('구름', 7, '갈색'))
구름 : 7 : 갈색 : 이상 없음
< undefined
```

객체 기본 매개변수 지정 방법

```
JS const test = function (object) { Untitled-1 ●  
1  const test = function (object) {  
2    // 과거(1)  
3    object.status = object.status !== undefined ? object.status : '이상 없음'  
4    // 과거(2)  
5    object.status = object.status ? object.status : '이상 없음'  
6    // 과거(3)  
7    object.status = object.status || '이상 없음'  
8  
9    // 현대(1)  
10   object = { status: '이상 없음', ...object }  
11   // 현대(2)  
12   fun = function ({name, age, color, status = '이상 없음'}) {  
13     |   return `${name} : ${age} : ${color} : ${status}`  
14   }  
}
```


1번째 방법

```
JS const dog = { Untitled-2 • JS const test = function (object)

1  const dog = {
2    name: '구름',
3    age: 7,
4    color: '갈색'
5  }
6
7  // (1)
8  console.log(dog.name)    // 구름
9  console.log(dog.age)     // 7
10 console.log(dog.color)   // 갈색
11
12 // (2)
13 console.log(dog.status)  // undefined
```

```
> const dog = {
    name: '구름',
    age: 7,
    color: '갈색'
}
```

// (1)

```
console.log(dog.name)    // 구름
console.log(dog.age)     // 7
console.log(dog.color)   // 갈색
```

// (2)

```
console.log(dog.status)  // undefined
```

구름

7

갈색

undefined

< undefined

과거 1, 2, 3의 예시

```
JS const dog = {  Untitled-2  •   JS const test = function (object) {  Untitled-1  •
1  const dog = {
2    name: '구름',
3    age: 7,
4    color: '갈색'
5  }
6
7  // 과거(1)
8  dog.status = dog.status !== undefined ? dog.status : '이상 없음'
9
10 // 과거(2)
11 dog.status = dog.status ? dog.status : '이상 없음'
12
13 // 과거(3)
14 dog.status = dog.status || '이상 없음'
15
```

2-3번의 경우 dog.status에 false로 변환되는 값이 오지 않을 것이라는 것이 확실해야 합니다! 그렇지 않으면 값을 넣었는데도 기본 값이 들어갑니다.

하아아 코드가 복잡해요
→ 조건부 연산자 + 짧은 조건문은 이때만 사용합니다. 외워주세요.

현대적인 방법 1 (전개연산자를 사용하는 코드)

```
JS const dog = { Untitled-2 • JS const test = function (c
1  const dog = {
2    name: '구름',
3    age: 7,
4    color: '갈색',
5    status: '다리가 아파요'
6  }
7
8  const newDog = {
9    status: '이상 없음',
10   ...dog
11 }
12
13 console.log(newDog)
```

```
> const dog = {
  name: '구름',
  age: 7,
  color: '갈색',
  status: '다리가 아파요'
}

const newDog = {
  status: '이상 없음',
  ...dog
}

console.log(newDog)
▶ {status: "다리가 아파요", name: "구름", age: 7, color: "갈색"}
```

현대적인 방법 2 (기본값을 지정하고 있는 코드)

```
JS const test = function ({  
1  const test = function ({  
2    name,  
3    age,  
4    color,  
5    status = '이상 없음'  
6  }) {  
7    return `${name} : ${age} : ${color} : ${status}`  
8  }  
9  
10 console.log(test(  
11   name: '구름',  
12   age: 7,  
13   color: '갈색'  
14 })))
```

```
> const test = function ({  
    name,  
    age,  
    color,  
    status = '이상 없음'  
  }) {  
    return `${name} : ${age} : ${color} : ${status}`  
  }  
  
  console.log(test(  
    name: '구름',  
    age: 7,  
    color: '갈색'  
  )))  
  
구름 : 7 : 갈색 : 이상 없음  
< undefined
```

현대적인 방법 1을 활용하여 (객체 속성 일괄 추출하기)

```
1  const test = function (object) {  
2    const {name, age, color, status} = { status: '이상 없음', ...object }  
3  
4    return `${object.name} : ${object.age} : ${object.color} : ${object.status}`  
5  }  
6  console.log(test(  
7    name: '구름',  
8    age: 7,  
9    color: '갈색'  
10  )))
```

```
1  const test = function (object) {  
2    const [ name, age, color, status ] = { status: '이상 없음', ...object }  
3  
4    return `${name} : ${age} : ${color} : ${status}`  
5  }  
6  console.log(test(  
7    name: '구름',  
8    age: 7,  
9    color: '갈색'  
10  )))
```


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

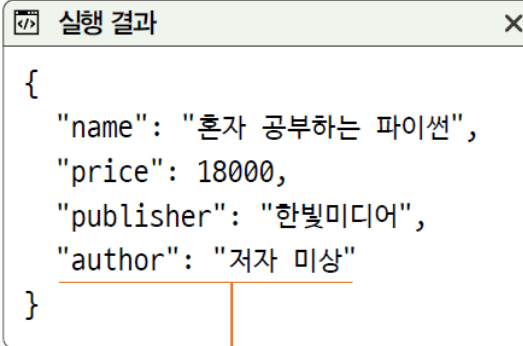
// 객체 내부에 속성이 있는지 확인합니다.

```
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
```

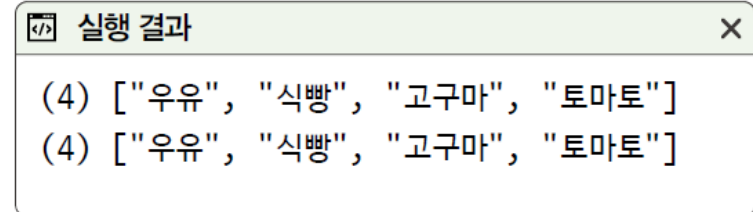
SECTION 6-3 객체와 배열 고급(6)

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 `undefined`가 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).

오늘도 고생하셨습니다.