

## 06. 객체

객체는 우리가 변수 혹은 상수를 사용하게 될 때 하나의 이름에 여러 종류의 값을 넣을 수 있게 해줍니다.

```
const dog = {  
  name: '멍멍이',  
  age: 2  
};  
  
console.log(dog.name);  
console.log(dog.age);
```

결과물은 다음과 같습니다.

```
멍멍이  
2
```

객체를 선언 할 때에는 이렇게 { } 문자 안에 원하는 값들을 넣어주면 됩니다. 값을 집어 넣을 때에는

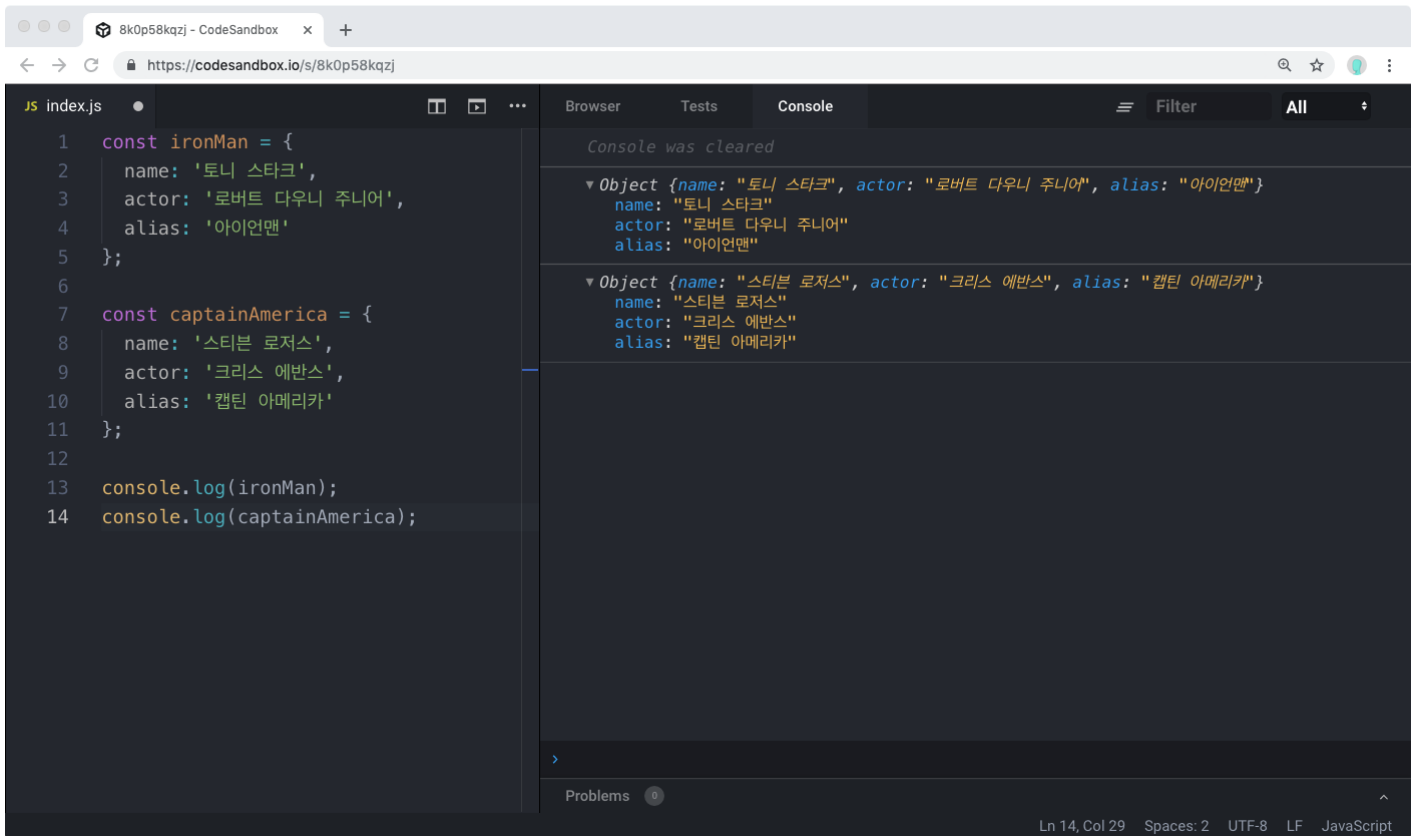
```
키: 원하는 값
```

형태로 넣으며, 키에 해당하는 부분은 공백이 없어야합니다. 만약에 공백이 있어야 하는 상황이라면 이를 따옴표로 감싸서 문자열로 넣어 주면 됩니다.

```
const sample = {  
  'key with space': true  
};
```

한번 영화 어벤져스의 캐릭터 아이언맨과 캡틴 아메리카의 정보를 객체안에 집어넣어봅시다.

```
const ironMan = {  
  name: '토니 스타크',  
  actor: '로버트 다우니 주니어',  
  alias: '아이언맨'  
};  
  
const captainAmerica = {  
  name: '스티븐 로저스',  
  actor: '크리스 에반스',  
  alias: '캡틴 아메리카'  
};  
  
console.log(ironMan);  
console.log(captainAmerica);
```



The screenshot shows a web browser window with the URL `https://codesandbox.io/s/8k0p58kqzj`. The editor displays a file named `index.js` with the following JavaScript code:

```
1 const ironMan = {
2   name: '토니 스타크',
3   actor: '로버트 다우니 주니어',
4   alias: '아이언맨'
5 };
6
7 const captainAmerica = {
8   name: '스티븐 로저스',
9   actor: '크리스 에반스',
10  alias: '캡틴 아메리카'
11 };
12
13 console.log(ironMan);
14 console.log(captainAmerica);
```

The console on the right shows the output of the code:

```
Console was cleared
▼ Object {name: "토니 스타크", actor: "로버트 다우니 주니어", alias: "아이언맨"}
  name: "토니 스타크"
  actor: "로버트 다우니 주니어"
  alias: "아이언맨"
▼ Object {name: "스티븐 로저스", actor: "크리스 에반스", alias: "캡틴 아메리카"}
  name: "스티븐 로저스"
  actor: "크리스 에반스"
  alias: "캡틴 아메리카"
```

결과물이 잘 출력 됐나요?

## 함수에서 객체를 파라미터로 받기

함수를 새로 만들어서 방금 만든 객체를 파라미터로 받아와서 사용해봅시다.

```
const ironMan = {
  name: '토니 스타크',
  actor: '로버트 다우니 주니어',
  alias: '아이언맨'
};

const captainAmerica = {
  name: '스티븐 로저스',
  actor: '크리스 에반스',
  alias: '캡틴 아메리카'
};

function print(hero) {
  const text = `${hero.alias}(${hero.name}) 역할을 맡은 배우는 ${
    hero.actor
  } 입니다.`;
  console.log(text);
}

print(ironMan);
print(captainAmerica);
```

결과물이 다음과 같이 나타나나요?

아이언맨(토니 스타크) 역할을 맡은 배우는 로버트 다우니 주니어 입니다.  
캡틴 아메리카(스티븐 로저스) 역할을 맡은 배우는 크리스 에반스 입니다.

위 코드에서는 줄이 너무 길어지는것을 방지하기 위하여 `${hero.actor}` 가 사용되는 부분에서 새 줄이 입력되었습니다. 이는 없어도 무방합니다만, 코드 샌드박스에서 저장시 자동으로 새 줄을 입력하게 됩니다. 이러한 기능을 방지하고 싶다면, 코드샌드박스 좌측의 설정 아이콘을 누르고 `.prettierrc` 를 열어서 `PrintWidth` 를 120 정도로 늘리시면 됩니다.

## 객체 비구조화 할당

print 함수를 보시면 파라미터로 받아온 hero 내부의 값을 조회 할 때 마다 hero. 를 입력하고 있는데, 객체 비구조화 할당이라는 문법을 사용하면 코드를 더욱 짧고 보기 좋게 작성 할 수 있습니다.

이 문법은 "객체 구조 분해" 라고 불리기도 합니다.

```
const ironMan = {
  name: '토니 스타크',
  actor: '로버트 다우니 주니어',
  alias: '아이언맨'
};

const captainAmerica = {
  name: '스티븐 로저스',
  actor: '크리스 에반스',
  alias: '캡틴 아메리카'
};

function print(hero) {
  const { alias, name, actor } = hero;
  const text = `${alias}(${name}) 역할을 맡은 배우는 ${actor} 입니다.`;
  console.log(text);
}

print(ironMan);
print(captainAmerica);
```

똑같은 결과가 나타났나요?

```
const { alias, name, actor } = hero;
```

이 코드가 객체에서 값들을 추출해서 새로운 상수로 선언해 주는 것 입니다.

여기서 더 나아가면, 파라미터 단계에서 객체 비구조화 할당을 할 수도 있습니다.

```
const ironMan = {
  name: '토니 스타크',
  actor: '로버트 다우니 주니어',
  alias: '아이언맨'
};

const captainAmerica = {
  name: '스티븐 로저스',
  actor: '크리스 에반스',
  alias: '캡틴 아메리카'
};

function print({ alias, name, actor }) {
  const text = `${alias}(${name}) 역할을 맡은 배우는 ${actor} 입니다.`;
  console.log(text);
}

print(ironMan);
print(captainAmerica);
```

어떨까요? 코드가 처음보다 훨씬 깔끔해졌지요?

## 객체 안에 함수 넣기

객체 안에 함수를 넣을 수도 있습니다. 한번 다음 코드를 실행해보세요.

```
const dog = {
  name: '멍멍이',
  sound: '멍멍!',
  say: function say() {
    console.log(this.sound);
  }
};

dog.say();
```

결과는 다음과 같습니다.

멍멍!

함수가 객체안에 들어가게 되면, `this` 는 자신이 속해있는 객체를 가르키게 됩니다.

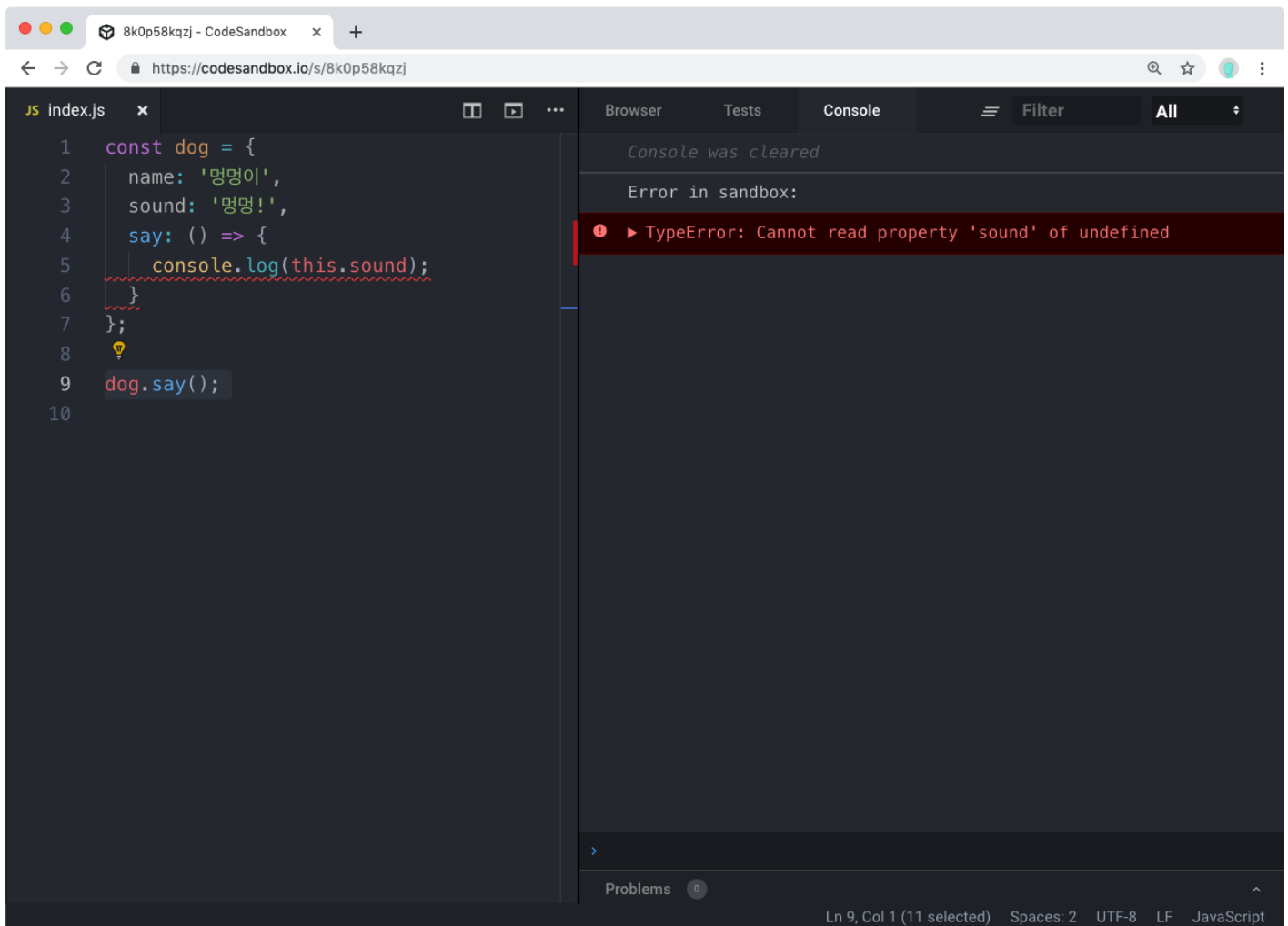
함수를 선언 할 때에는 이름이 없어도 됩니다.

```
const dog = {
  name: '멍멍이',
  sound: '멍멍!',
  say: function() {
    console.log(this.sound);
  }
};

dog.say();
```

이전과 똑같이 작동 할 것입니다.

객체 안에 함수를 넣을 때, 화살표 함수로 선언한다면 제대로 작동하지 않습니다.



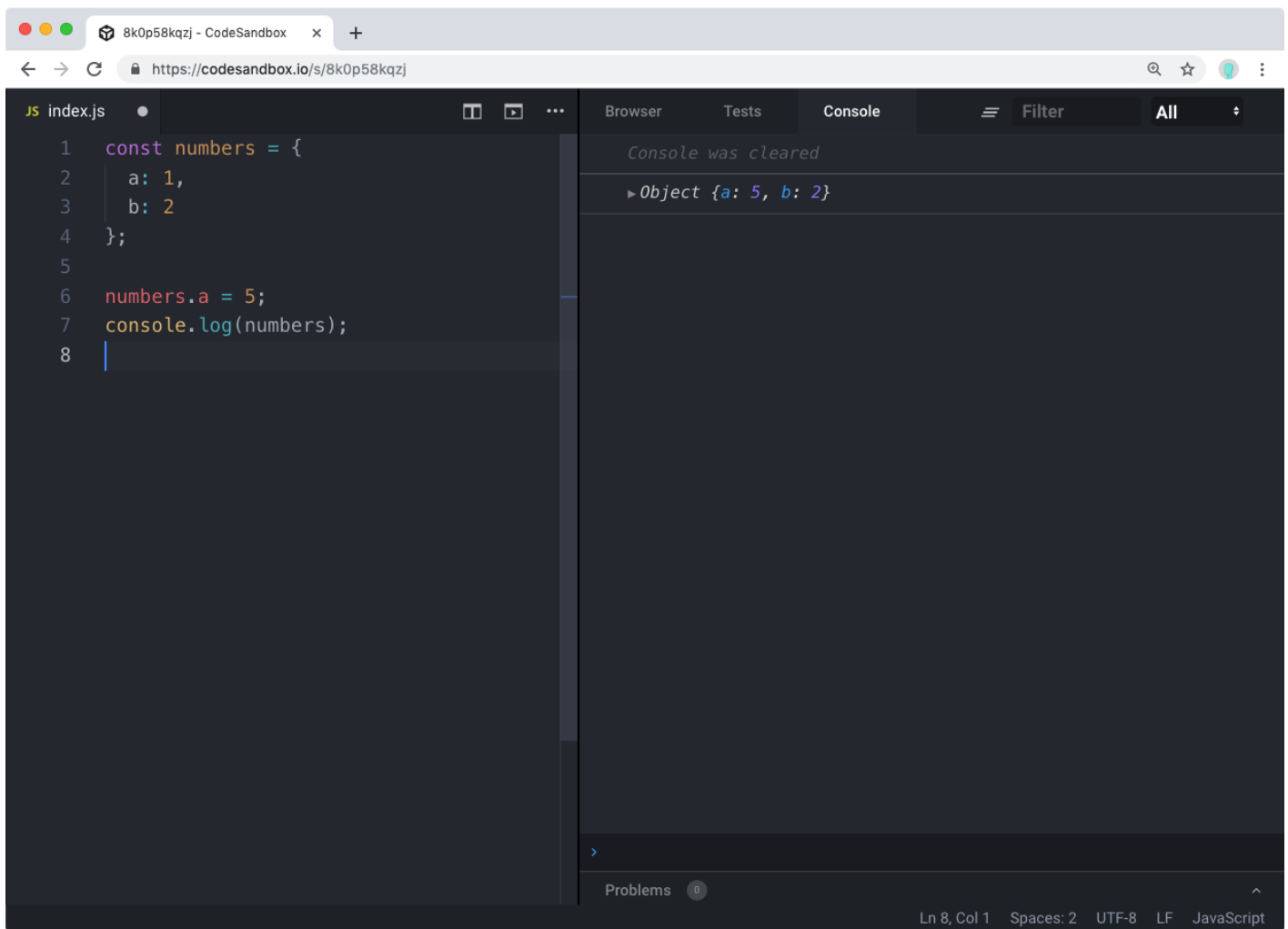
이유는, `function` 으로 선언한 함수는 `this` 가 제대로 자신이 속한 객체를 가르키게 되는데, 화살표 함수는 그렇지 않기 때문입니다.

## Getter 함수와 Setter 함수

객체 안에 `Getter` 함수와 `Setter` 함수를 설정하는 방법을 알아보시다. 객체를 만들고 나면, 다음과 같이 객체안의 값을 수정 할 수도 있는데요,

```
const numbers = {
  a: 1,
  b: 2
};

numbers.a = 5;
console.log(numbers);
```



Getter 함수와 Setter 함수를 사용하게 되면 특정 값을 바꾸려고 하거나, 특정 값을 조회하려고 할 때 우리가 원하는 코드를 실행 시킬 수 있습니다.

다음 코드를 따라 적어보세요.

```
const numbers = {
  a: 1,
  b: 2,
  get sum() {
    console.log('sum 함수가 실행됩니다!');
    return this.a + this.b;
  }
};

console.log(numbers.sum);
numbers.b = 5;
console.log(numbers.sum);
```

```
1  const numbers = {
2    a: 1,
3    b: 2,
4    get sum() {
5      console.log('sum 함수가 실행됩니다!');
6      return this.a + this.b;
7    }
8  };
9
10 console.log(numbers.sum);
11 numbers.b = 5;
12 console.log(numbers.sum);
13
```

The console output shows the following sequence of events:

- Console was cleared
- sum 함수가 실행됩니다!
- 3
- sum 함수가 실행됩니다!
- 6

At the bottom of the editor, the status bar indicates: Ln 12, Col 26, Spaces: 2, UTF-8, LF, JavaScript.

우리는 `numbers.sum()` 을 한 것이 아니라 `number.sum` 을 조회했을 뿐인데, 함수가 실행되고 그 결과값이 출력되었습니다. 이런식으로 Getter 함수는 특정 값을 조회 할 때 우리가 설정한 함수로 연산된 값을 반환합니다. 이번에는 Setter 함수를 사용해봅시다.

```

const numbers = {
  _a: 1,
  _b: 2,
  sum: 3,
  calculate() {
    console.log('calculate');
    this.sum = this._a + this._b;
  },
  get a() {
    return this._a;
  },
  get b() {
    return this._b;
  },
  set a(value) {
    console.log('a가 바뀝니다.');
    this._a = value;
    this.calculate();
  },
  set b(value) {
    console.log('b가 바뀝니다.');
    this._b = value;
    this.calculate();
  }
};

console.log(numbers.sum);
numbers.a = 5;
numbers.b = 7;
numbers.a = 9;
console.log(numbers.sum);
console.log(numbers.sum);
console.log(numbers.sum);

```

The screenshot shows a web browser window with the URL `https://codesandbox.io/s/8k0p58kqzj`. The code editor on the left contains the same JavaScript code as shown in the previous block. The right-hand side of the interface shows the 'Console' tab, which displays the following output:

- `Console was cleared`
- `3`
- `a가 바뀝니다.`
- `calculate`
- `b가 바뀝니다.`
- `calculate`
- `a가 바뀝니다.`
- `calculate`
- `16`
- `16`
- `16`

The status bar at the bottom indicates the file is `index.js`, line 16, column 30, with 2 spaces, UTF-8 encoding, LF line endings, and JavaScript syntax.

Setter 함수를 설정 할 때에는 함수의 앞부분에 `set` 키워드를 붙입니다.

Setter 함수를 설정하고 나면, `numbers.a = 5` 이렇게 값을 설정했을 때 5 를 함수의 파라미터로 받아오게 됩니다. 위 코드에서는 객체 안에 `_a`, `_b` 라는 숫자를 선언해주고, 이 값들을 위한 Getter 와 Setter 함수를 설정해주었습니다.

아까 전에는 만든 객체에서는 `numbers.sum` 이 조회 될 때마다 덧셈이 이루어졌었지만, 이제는 `a` 혹은 `b` 값이 바뀔 때마다 `sum` 값을 연산합니다