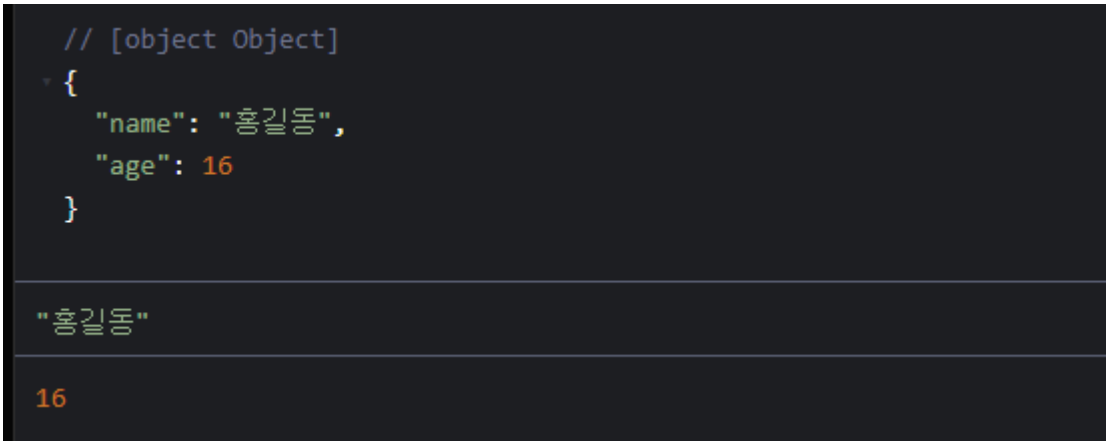
객체

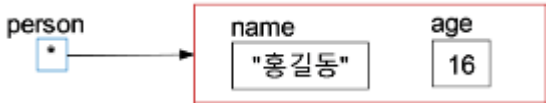
객체 생성

```
let person = { name: "홍길동", age: 16 };

console.log(person); //person 변수가 참조하는 객체를 출력한다.
console.log(person.name); //person 변수가 참조하는 객체 내부의 name 멤버 변수 값을 출력한다.
console.log(person.age);
```



`let person = { name: "홍길동", age: 16 };`

객체를 생성하고, 그 객체에 대한 참조를 person 변수에 대입한다.
그 객체 내부에 name 멤버 변수와 age 멤버 변수가 만들어진다.
name 멤버 변수에 "홍길동" 문자열이 대입된다.
age 멤버 변수에 16 을 대입된다.



참조형 (reference type)

객체는 참조형이다.
객체 변수에는 그 객체에 대한 참조(reference)만 저장된다.

기본 자료형 (primitive type)

기본 자료형은 값형이다. (value type)
기본 자료형 변수에는 값이 저장된다.

> 💡 Java에서 String은 객체이고 참조형이다.
> Javascript에서 문자열은 기본 자료형이고 값형이다..

```
let person1 = { };

person1.name = "홍길동";
person1.age = 16;
console.log(person1);

let person2 = { name: "임꺽정" };
person2.age = 19;
console.log(person2);
```

```
// [object Object]
{
  "name": "홍길동",
  "age": 16
}
```

```
// [object Object]
{
  "name": "임꺽정",
  "age": 19
}
```

`let person1 = { };`
객체를 생성하고, 그 객체에 대한 참조를 person1 변수에 대입한다.
그 객체 내부에 멤버 변수가 없다.

`person1.name = "홍길동";`
person1 변수가 참조하는 객체에, name 맴버 변수가 있다면, 그 변수에 "홍길동" 문자열을

대입한다.

name 멤버 변수가 없다면 만들고, 그 변수에 "홍길동" 문자열을 대입한다.

`person1.age = 16;`
person1 변수가 참조하는 객체에, age 맴버 변수가 있다면, 그 변수에 16을 대입한다.
agee 멤버 변수가 없다면 만들고, 그 변수에 16을 대입한다.

`let person2 = { name: "임꺽정" }`
객체를 생성하고, 그 객체에 대한 참조를 person2 변수에 대입한다.
그 객체 내부에 name 멤버가 만들어진다.
name 멤버 변수에 "임꺽정" 문자열이 대입된다.

`person2.age = 19;`
person2 변수가 참조하는 객체에, age 맴버 변수가 있다면, 그 변수에 19 를 대입한다.
agee 멤버 변수가 없다면 만들고, 그 변수에 19 를 대입한다.

```
let person = { };

person["name"] = "홍길동";
person["age"] = 16;

console.log(person["name"]);
console.log(person["age"]);
```

```
"홍길동"
16
```

`person["name"] = "홍길동";`
위 코드는 아래 코드와 동일하다.

`person.name = "홍길동";`

`person["age"] = 16;`
위 코드는 아래 코드와 동일하다.

`person.age = 16;`

`console.log(person["name"]);`
위 코드는 아래 코드와 동일하다.

`console.log(person.name);`

`console.log(person["age"]);`
위 코드는 아래 코드와 동일하다.

`console.log(person.age);`

```
function createPerson(s, i) {
  return { name: s, age: i };
}

let person1 = createPerson("홍길동", 16);
let person2 = createPerson("임꺽정", 18);

console.log(person1);
console.log(person2);
```

```
// [object Object]
{
  "name": "홍길동",
  "age": 16
}
```

```
// [object Object]
{
  "name": "임꺽정",
  "age": 18
}
```

`function createPerson(s, i) {`
`  return { name: s, age: i };`
`}`

createPerson 함수는 객체를 생성하여 리턴한다.

그 객체 내부에는 name 멤버 변수와 age 멤버 변수가 만들어진다.
name 멤버 변수에 s 파라미터 변수의 값이 대입되고,
age 멤버 변수에 i 파라미터 변수의 값이 대입된다.

`let person1 = createPerson("홍길동", 16);`
createPerson 함수가 리턴하는 객체에 대한 참조를 person1 변수에 대입한다.

```
function createPerson() {
  return { name: "홍길동", age: 16 }
}

let person1 = createPerson()
let person2 = createPerson()

person1.name = "임꺽정"
person2.age = 20
console.log(person1)
console.log(person2)
```

```
// [object Object]
{
  "name": "임꺽정",
  "age": 16
}
```

```
// [object Object]
{
  "name": "홍길동",
  "age": 20
}
```

createPerson 함수는 호출될 때 마다 새 객체를 생성해서 리턴한다.
그래서 person1 변수와 person2 변수가 참조하는 객체는 서로 다른 두 객체이다.

객체 비교

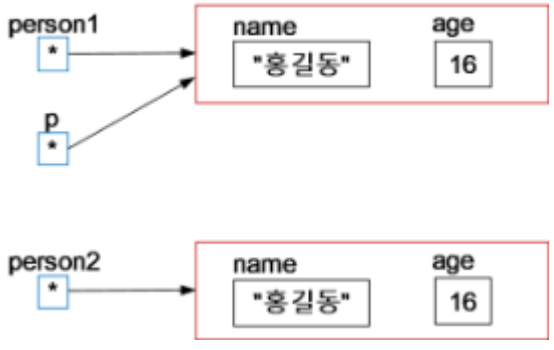
```
let person1 = { name: "홍길동", age: 16 };
let person2 = { name: "홍길동", age: 16 };
let p = person1;

console.log(person1 == person2);
console.log(person1 == p);
```

```
false
true
```

`let person1 = { name: "홍길동", age: 16 };`
객체를 한 개 생성하고, 그 객체에 대한 참조를 person1 변수에 대입한다.

`let person2 = { name: "홍길동", age: 16 };`
또 객체를 한 개 생성하고, 그 객체에 대한 참조를 person2 변수에 대입한다.

`let p = person1;`
person1 변수의 값을 변수 p에 대입한다.
person1 변수의 값은 객체에 대한 참조이다. 그 참조를 변수 p에 대입한다.



참조 비교 (== 연산자)

참조형인 객체를 == 연산자로 비교하면, 객체 내부 값을 비교하지 않고,
두 참조를 비교한다. (두 참조가 동일한 객체를 참조하는지 비교한다.)

위 그림에서
person1 변수에 p 변수는 같은 객체를 참조하고 있다.
따라서 person1 == p 표현식의 값은 true 이다.

person1 변수와 person2 변수는 서로 다른 객체를 참조하고 있다.
따라서 person1 == person2 표현식의 값은 false 이다.

참조를 비교하는 것은 identity를 비교하는 것이다.
내부 값을 비교하는 것은 equality를 비교하는 것이다.

Java

```
String a = "hello";
String b = "hello";

a == b  // java 에서 false
```

Java에서 String은 객체 즉 참조형 (reference type)
a, b 변수에 참조 주소가 들어있고, 두 String 객체의 주소가 서로 다르다.

Javascript

```
let a = "hello";
let b = "hello";

a == b // javascript 에서 true
```

Javascript에서 문자열(string)은 기본 자료형 (value type)
a, b 변수에 문자열이 들어있고, 변수의 내용이 같다.

```
let s1 = "hello world";
let s2 = "hello " + "world";

console.log(s1 == s2); //true
```

javascript에서 문자열은 기본 자료형이다.
기본 자료형을 == 연산자로 비교하면, 값이 같은지를 비교한다.

s1 변수의 값과 s2 변수의 값이 같으므로, s1 == s2 표현식의 값은 true 이다.

```
let person1 = { name: "홍길동", age: 16 };
let person2 = { name: "홍길동", age: 16 };

function equals(p1, p2) {
  return p1.name == p2.name &&
         p1.age == p2.age;
}

console.log(equals(person1, person2)); //true
```

객체의 내부 값을 비교하려면, 위와 같은 비교 함수를 구현해야 한다.

java에서는 이름이 같은 메소드가 여러 개일 수 있다. (method overloading)
javscript에서는 이름이 같은 함수가 여러 개일 수 없다.

객체 배열

```
let person1 = { name: "홍길동", age: 16 };
let person2 = { name: "임꺽정", age: 18 };
let person3 = { name: "전우치", age: 19 };
let persons = [ person1, person2, person3 ];
```

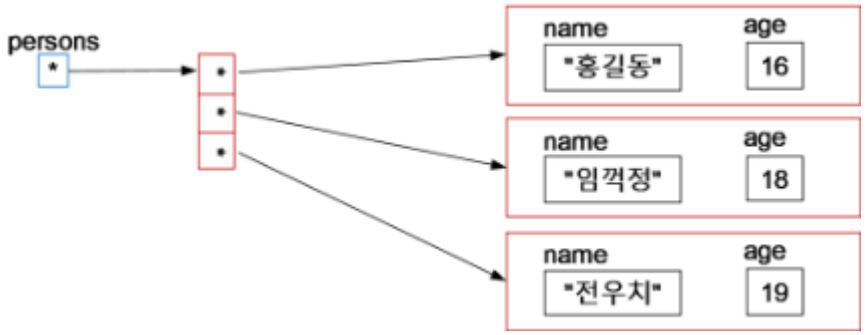
```
console.log(persons);
```

```
// [object Array] (3)
[// [object Object]
{
  "name": "홍길동",
  "age": 16
},// [object Object]
{
  "name": "임꺽정",
  "age": 18
},// [object Object]
{
  "name": "전우치",
  "age": 19
}]
```

배열도 참조형이다.
배열 변수에는 배열에 대한 참조만 저장된다.

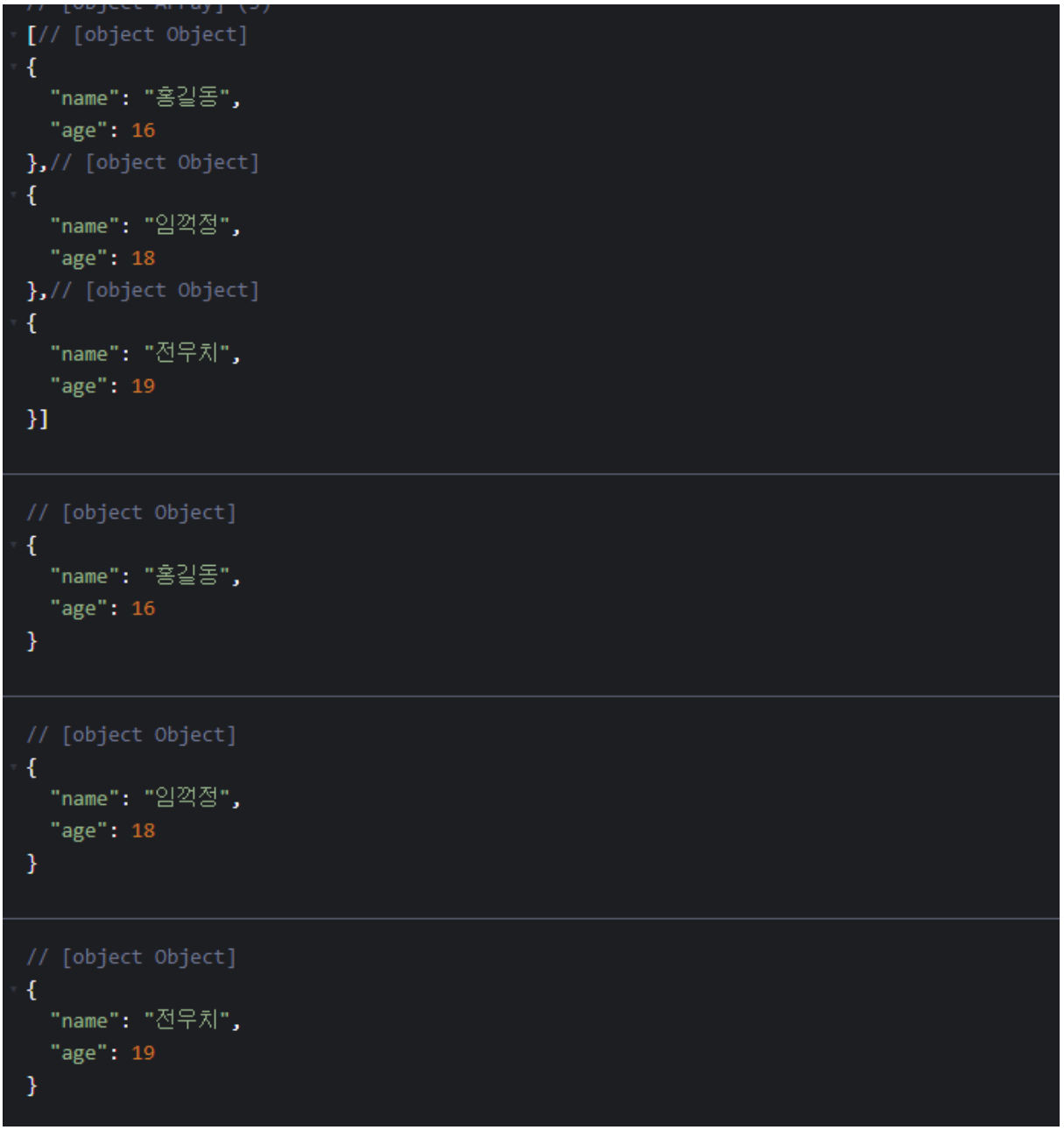
`let persons = [ person1, person2, person3 ];`
배열이 생성되고, 이 배열에 대한 참조가 persons 변수에 대입된다.
persons[0]에 person1 변수의 값이 대입된다.

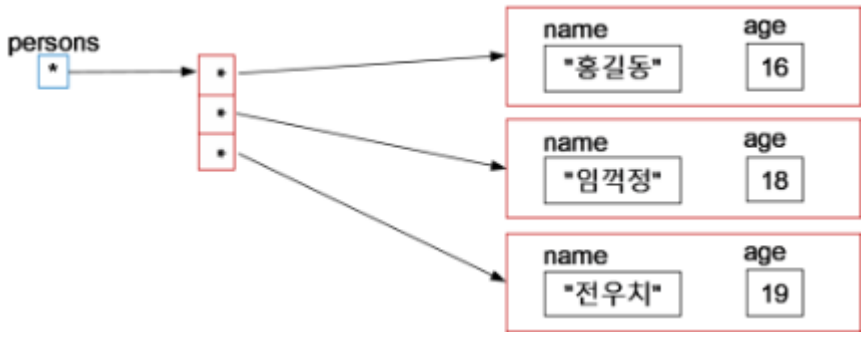
즉 person1 변수가 참조하는 객체에 대한 참조가 대입된다.
마찬가지로 person[1], person[2]에도, person2, person3 변수의 값이 각각 대입된다.




```
let persons = [
  { name: "홍길동", age: 16 },
```

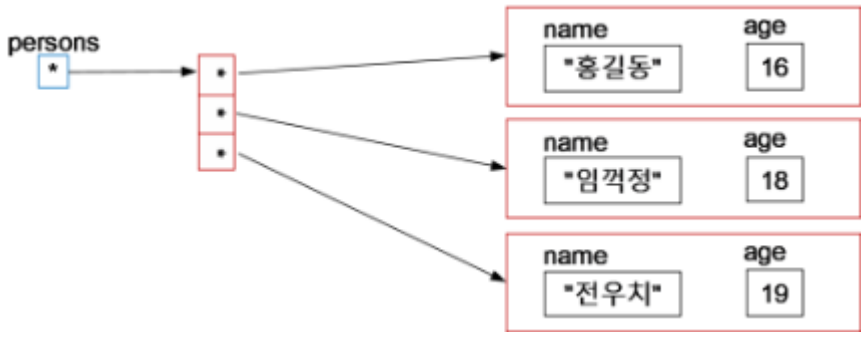
```
  { name: "임꺽정", age: 18 },
  { name: "전우치", age: 19 }
];

console.log(persons);

for (let i = 0; i < persons.length; ++i)
  console.log(persons[i]);
```



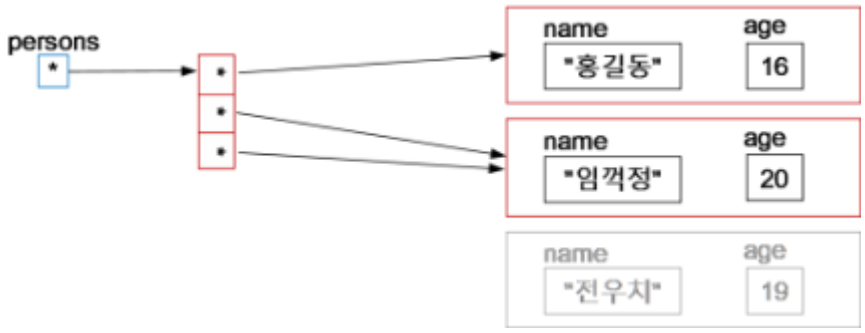



```
let persons = [
  { name: "홍길동", age: 16 },
  { name: "임꺽정", age: 18 },
  { name: "전우치", age: 19 }
];
console.log(persons);

persons[2] = persons[1];
persons[1].age = 20;
console.log(persons);
```

```
let persons = [
 { name: "홍길동", age: 16 },
 { name: "임꺽정", age: 18 },
 { name: "전우치", age: 19 }
];
```




```
persons[2] = persons[1];
persons[1].age = 20;
```




```
// [object Array] (3)
[// [object Object]
{
  "name": "홍길동",
  "age": 16
},// [object Object]
{
  "name": "임꺽정",
  "age": 20
},// [object Object]
{
  "name": "임꺽정",
  "age": 20
}]
```

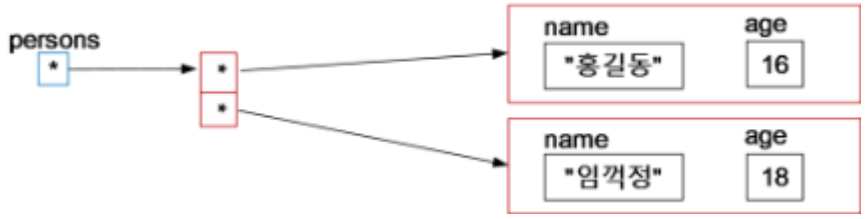
```
let persons = [
  { name: "홍길동", age: 16 },
  { name: "임꺽정", age: 18 }
];
console.log(persons);

persons = [
  { name: "전우치", age: 19 },
  { name: "이몽룡", age: 16 }
];
console.log(persons);
```

```
[// [object Object]
{
  "name": "홍길동",
  "age": 16
},// [object Object]
{
  "name": "임꺽정",
  "age": 18
}]
```

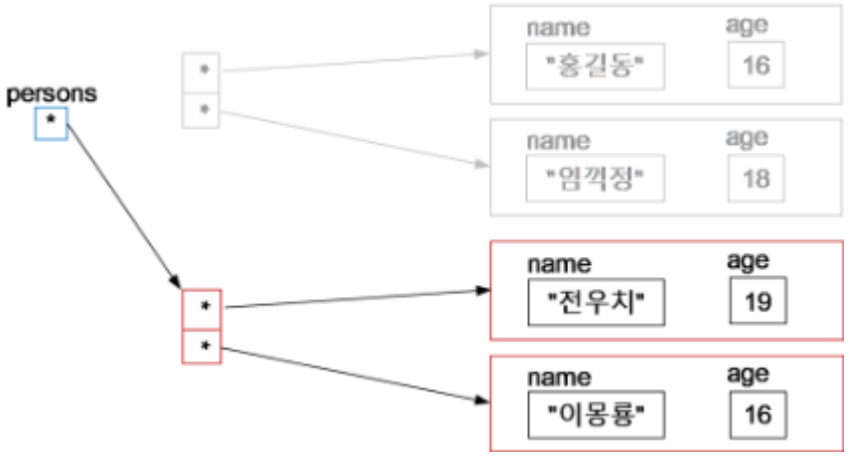


```
// [object Array] (2)
[// [object Object]
{
  "name": "전우치",
  "age": 19
},// [object Object]
{
  "name": "이몽룡",
  "age": 16
}]
```


```
let persons = [
  { name: "홍길동", age: 16 },
  { name: "임꺽정", age: 18 }
];
```




```
persons = [
  { name: "전우치", age: 19 },
  { name: "이몽룡", age: 16 }
];
```



메소드

```
let rectangle = {
  width: 5,
  height: 7,
  area: function() { return this.width * this.height; }
};

console.log(rectangle.area()); //35
```

```
let rectangle = {
  width: 5,
  height: 7,
  area: function() { return this.width * this.height; }
};
```

객체가 한 개 생성되고, 그 객체에 대한 참조가 rectangle 변수에 대입된다.
이 객체에는 width, height, area 멤버 변수가 만들어진다.
width 멤버 변수에 5가 대입되고, height 멤버 변수에 7이 대입된다.
area 멤버 변수에 함수가 대입된다.

객체의 멤버 변수에 대입된 함수는, 그 객체의 메소드가 된다.

`rectangle.area()`
rectangle 객체의 area 멤버 변수의 값인 함수를 호출한다.
이때 이 함수 본문에서 this는 rectangle 객체를 참조한다.

`console.log(rectangle.area());`
rectangle 객체의 area 메소드를 호출하고, 그 리턴값을 출력한다.

> 💡 java 메소드에서 this는 생략 가능하다.
> javascript 메소드에서 this는 생략할 수 없다.

```
let rectangle = {
  width: 5,
  height: 7
};

rectangle.area = function () {
  return this.width * this.height;
}

console.log(rectangle.area()); //35
```

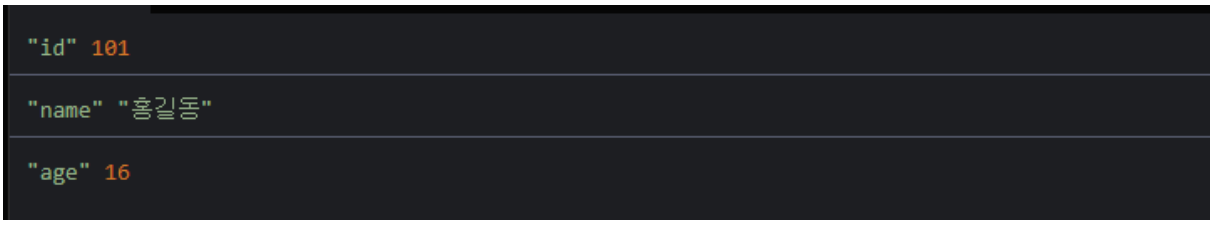
```
rectangle.area = function () {
  return this.width * this.height;
}
```

reactangle 객체에 area 멤버 변수가 있다면, 그 멤버 변수에 함수를 대입한다.
area 멤버 변수가 없다면 만들고, 그 멤버 변수에 함수를 대입한다.

객체의 멤버 변수에 대입된 함수는, 그 객체의 메소드가 된다.

속성 탐색(for in문)

```
let person = { id: 101, name: "홍길동", age: 16 };
//객체에 for in문을 사용하면 key값으로 속성명이 문자열로 꺼내어진다.
for (let key in person) {
  let value = person[key];
  console.log(key, value);
}
```



`for (let key in person) {`
person 객체의 멤버 변수 각각에 대해서, 그 멤버 변수 이름 문자열이 key 변수에 대입 되고 반복문이 실행된다.

person 객체의 멤버 변수는 3개 이므로, 위 for 문은 3번 반복한다.
첫째 반복에서 key 변수에 "id" 문자열이 대입 된다.
둘째 반복에서 key 변수에 "name" 문자열이 대입 된다.
셋째 반복에서 key 변수에 "age" 문자열이 대입 된다.

`let value = person[key];`
key 변수의 값이 "id" 이면, person 객체의 id 멤버 변수 값이 value 변수에 대입된다.
key 변수의 값이 "name" 이면, person 객체의 name 멤버 변수 값이 value 변수에 대입 된다.
key 변수의 값이 "age" 이면, person 객체의 age 멤버 변수 값이 value 변수에 대입된다.

`let value = person.name;`
위 코드와 아래 코드는 동일하다.

`let value = person["name"];`
위 코드와 아래 코드는 동일하다.

```
let key = "name";
let value = person[key];
```

```
let rectangle = {
  width: 5,
  height: 7,
  area : function() { return this.width * this.height; }
};

for (let key in rectangle) {
  let value = rectangle[key];
  console.log(key, value);
}
```

```
"width" 5
"height" 7
 "area" function () {return this.width * this.height;}
```

rectangle 객체 내부의 멤버 변수는 세 개이다. (width, height, area)
area 멤버 변수의 값은 함수이다.

멤버 변수 제거

```
let person = { name: "홍길동", age: 16, department: "소프" };
console.log(person);

delete person.department;
console.log(person);
```

```
// [object Object]
{
  "name": "홍길동",
  "age": 16,
  "department": "소프"
}
```

```
// [object Object]
{
  "name": "홍길동",
  "age": 16
}
```

`delete person.department;`
person 객체의 department 멤버 변수를 제거한다.

구조 분해 할당 (destructureing assignment)

```
let person = { name: "홍길동", age: 16 };
let { name, age } = person;
console.log(name, age); //홍길동 16
```

`let { name, age } = person;`
name, age 지역 변수가 생성된다.

name 지역 변수에 person.name 값이 대입되고,
age 지역 변수에 person.age 값이 대입된다.
지역 변수의 순서는 중요하지 않다. 지역 변수 이름이 객체의 속성명과 같아야 한다.

```
let person = { name: "홍길동", age: 16, english: 90, math: 8
5, history: 95 };
let { name, age, ...scores } = person;
console.log(name, age);
console.log(scores);
```

```
"홍길동" 16

// [object Object]
{
  "english": 90,
  "math": 85,
  "history": 95
}
```

`let { name, age, ...scores } = person;`

name, age, scores 지역 변수가 생성된다.

name 지역 변수에 person.name 값이 대입되고,
age 지역 변수에 person.age 값이 대입된다.
scores 지역 변수에 객체가 대입된다.
person 객체의 속성 중에서, name 속성과 age 속성을 제외한 나머지 속성들이 scores 객체에 복사된다.

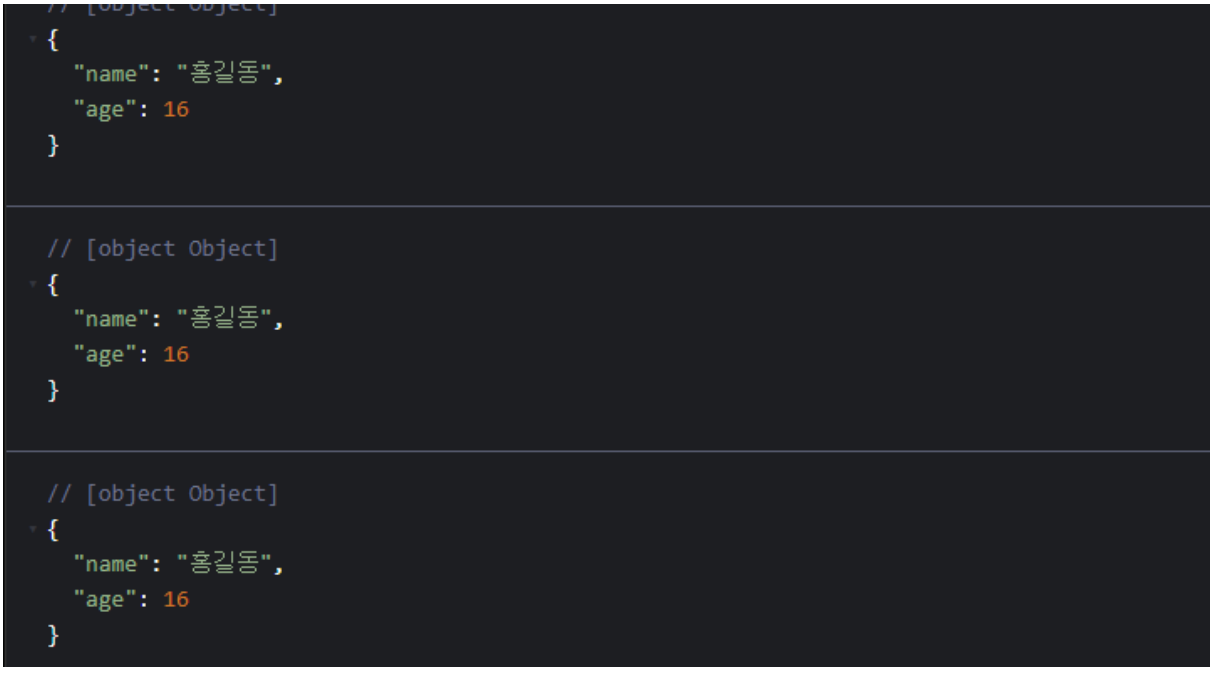
위 코드에서 name, age 속성이나 다른 속성들의 순서는 중요하지 않다.
예를 들어 아래처럼 수정하고 실행해 보자.

```
let person = { id: 3, name: "홍길동", english: 90, math: 85,
history: 95, age: 16 };
let { name, age, ...scores } = person;
console.log(name, age);
console.log(scores);
```

id 속성이 추가되었고, age 속성은 뒤로 이동하였다.

```
"홍길동" 16

// [object Object]
{
  "id": 3,
  "english": 90,
  "math": 85,
  "history": 95
}
```

```js
let person1 = { name: "홍길동", age: 16 };
console.log(person1);

let name = "홍길동";
let age = 16;

let person2 = { name: name, age: age };
console.log(person2);

//속성명과 변수명이 같으면 생략할 수 있다
let person3 = { name, age };
console.log(person3);
```



위 코드에서 person1, person2, person3 변수에 대입되는 객체는 모두 동일한 값이다.

`let person2 = { name: name, age: age };`
person2 지역 변수에 객체가 대입된다.
이 객체에는 name 속성과 age 속성이 만들어진다. (위 코드에서 name: 부분과 age: 부분은 속성의 이름이다)
이 속성들의 값은 각각 name, age 지역 변수의 값이다.

```
let person3 = { name, age };
```

만들어야 할 속성의 이름이 지역 변수의 이름과 같을 경우에,
이름을 생략할 수 있다.
