# JS-중급 12

| | |
|---|---|
| 태그 | |
| 날짜 | @2023년 6월 5일 |

## 상속, prototype

```
> const user = {
      name : 'Mike'
  }
< undefined
> user.name
< "Mike"
>
```

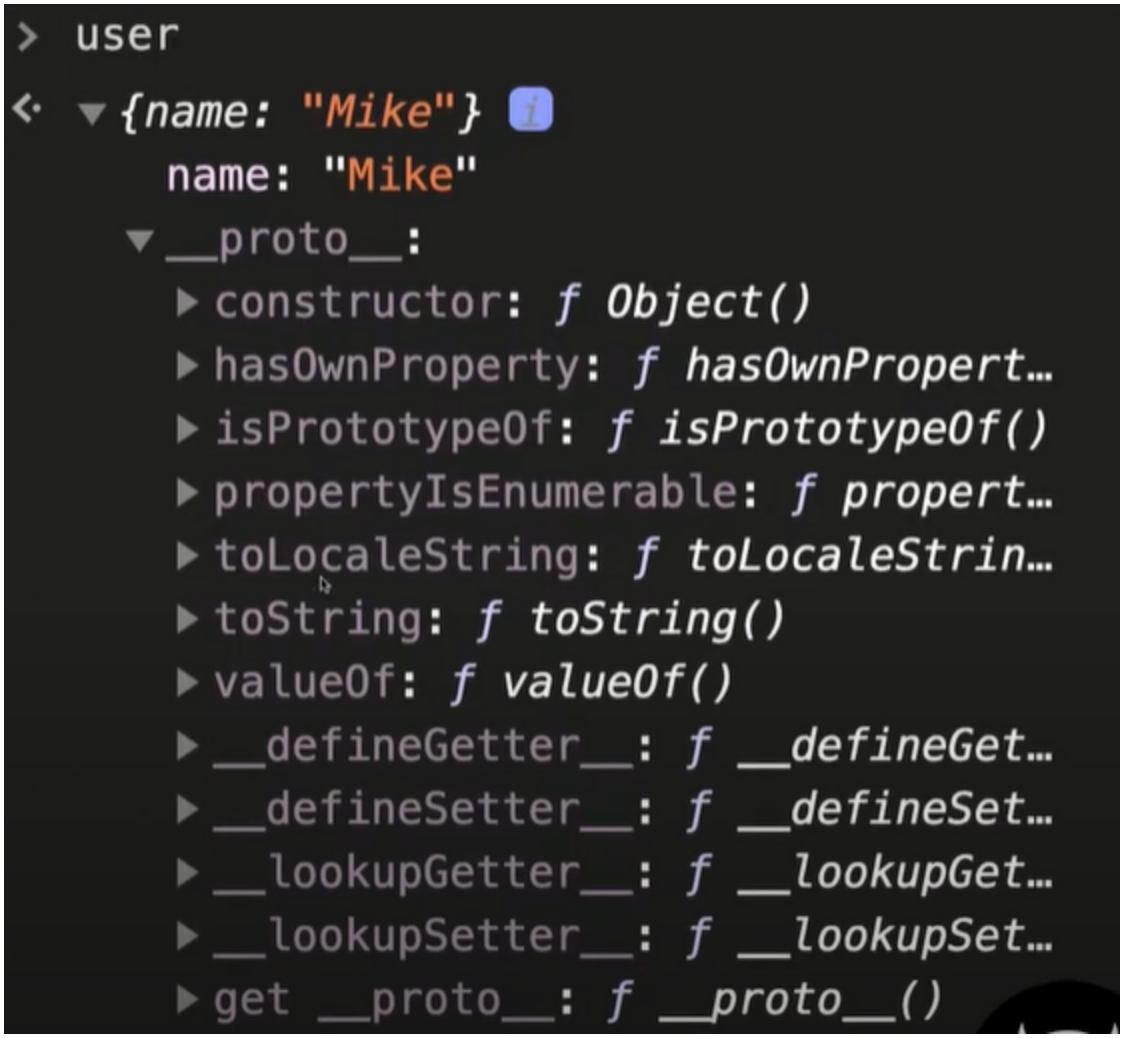
객체에는 자신이 프로퍼티를 가지고 있는지 확인하는 메소드가 있는데

hasOwnProperty 이다.

```
> user.hasOwnProperty('name')
< true
> user.hasOwnProperty('age');
< false
```

유저객체 내에 있으면 true 없으면 false

근데 만든적이 없는데 어디있는 걸까?




여기에 있따.

__proto__ : 이것을 프로토 타입이라고 한다.

객체에서 프로퍼티를 읽으려고 하는데 없으면 여기서 찾게 된다.

만약 hasOwnProperty가 있으면 어떻게 될까

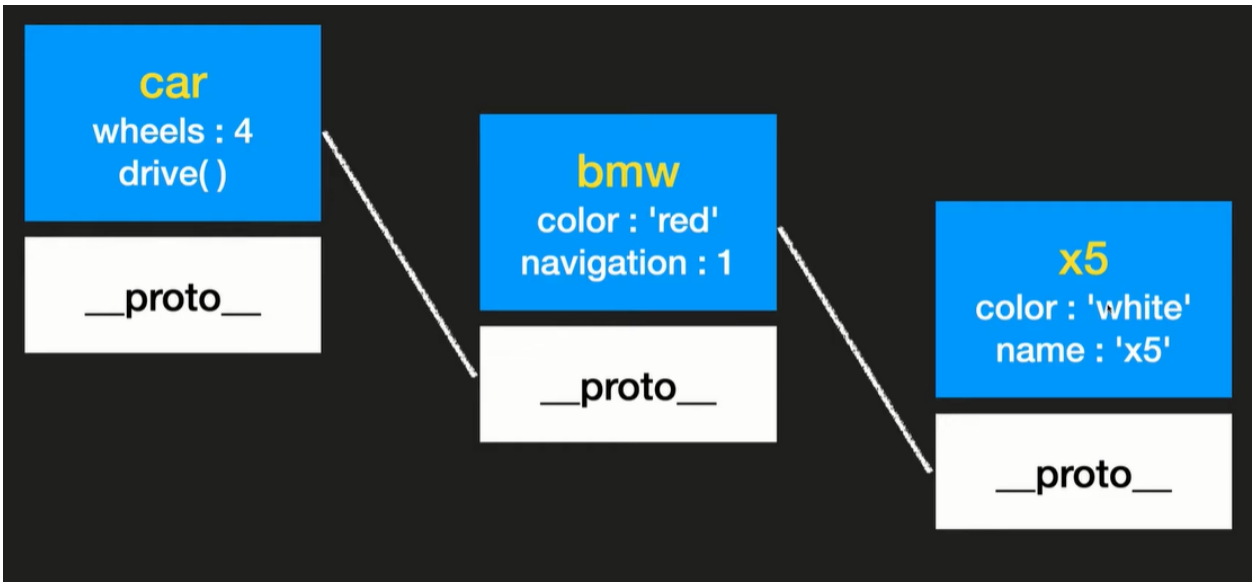

지금 만든 메소드가 동작한다.

일단 그 객체에 프로퍼티가 있으면 거기서 멈춘다. 그 후 없으면 프로토타입에서 찾는다.

```
const bmw = {
  color : "red",
  wheels : 4,
  navigation : 1,
  drive() {
    console.log("drive...");
  },
};
```

```
const benz = {
  color : "black",
  wheels : 4,
  drive() {
    console.log("drive...");
  },
};

const audi = {
  color : "blue",
  wheels : 4,
  drive() {
    console.log("drive...")
  },
};

// 상속이라는 개념을 이용
// 저렇게 차들이 늘어나면 새로운 변수로 만들어지는데
// 공통된 부분은 어떻게 처리하게 될까.

const car = {
  wheels : 4,
  drive(){
    console.log("drive...");
  },
};

const bmw = {
  color : "red",
  wheels : 4,
  navigation : 1,
  drive() {
    console.log("drive...");
  },
};

const benz = {
  color : "black",
  wheels : 4,
  drive() {
    console.log("drive...");
  },
};

const audi = {
  color : "blue",
  wheels : 4,
  drive() {
    console.log("drive...")
  },
};

// car라는 상위 개념의 객체를 만든다.
// 그 후 만들었던 자동차에서 wheels와 drive는 지운다.
```

```
const car = {
  wheels : 4,
  drive(){
    console.log("drive...");
  },
};

const bmw = {
  color : "red",
  navigation : 1,
 };

const benz = {
  color : "black",
};

const audi = {
  color : "blue",
};

bmw.__proto__ = car;
benz.__proto__ = car;
audi.__proto__ = car;

// car 가 bmw의 prototype이 된다.
// 다르게 말하면 bmw는 car의 상속을 받는다
bmw.wheels
4
audi.drive
drive() {
    console.log("drive...");
  }


// 상속은 이어질 수 있다.
const x5 = {
  color : "white",
  name : "x5",
}

x5.__proto__ = bmw;
```




prototype chain 이라고 한다.

```
for(p in x5){
console.log(p);
}
```

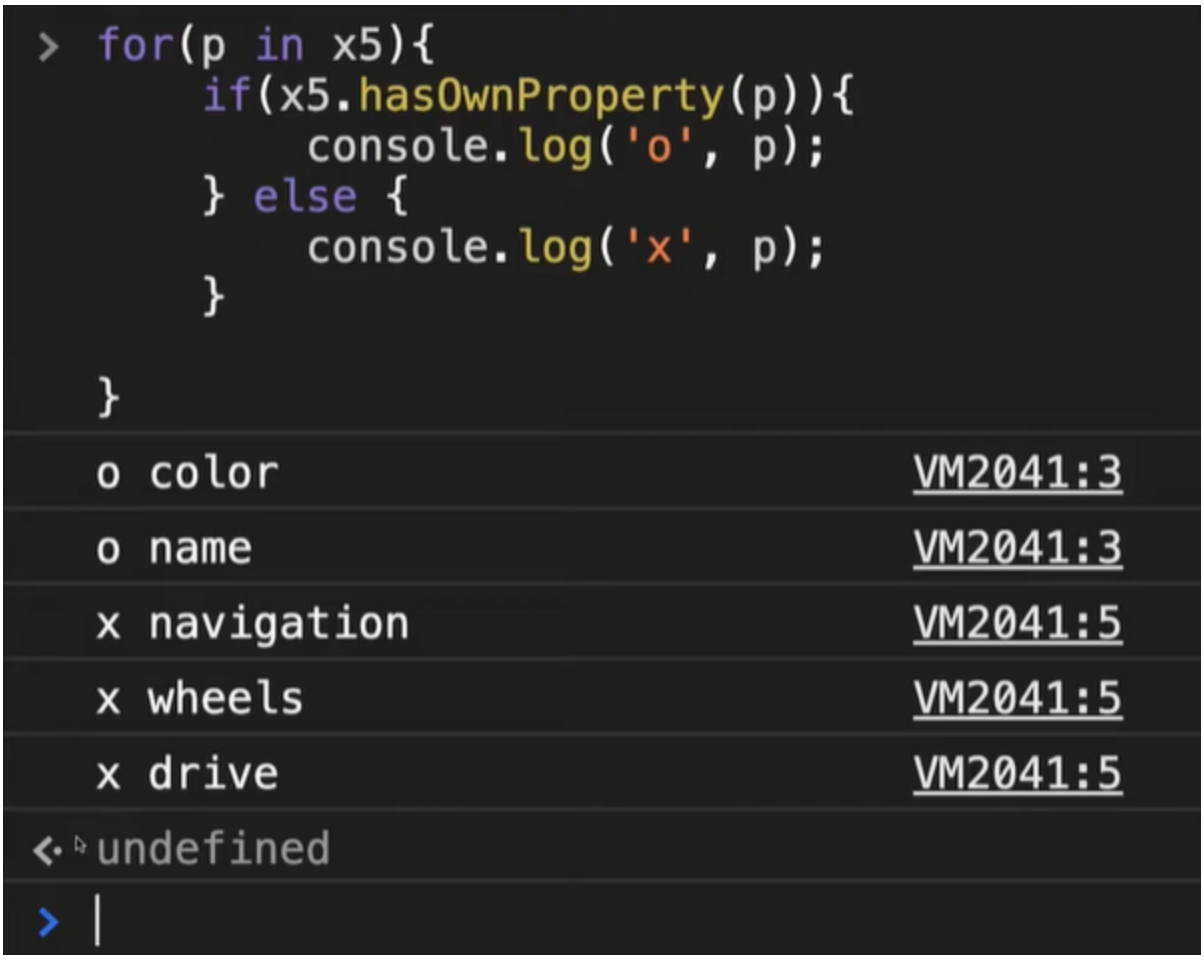
```
"color"
"name"
"navigation"
"wheels"
"drive"
undefined
```

color 와 name을 제외하고는 모두 상속받은 것들

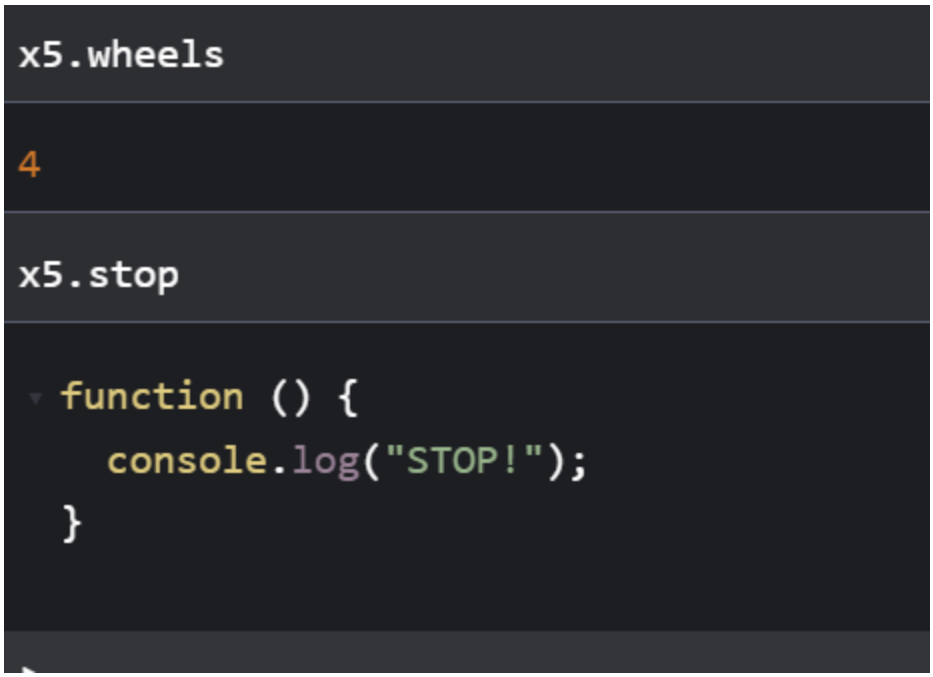
```
Object.keys(x5)

// [object Array] (2)
["color","name"]

Object.values(x5)

// [object Array] (2)
["white","x5"]
```

키, 값 과 관련된 객체 내장 메소드는 상속된 프로퍼티는 나오지 않는다.

만약 for in 문에서 사용하고 싶다면 hasOwnProperty를 사용하면 된다.

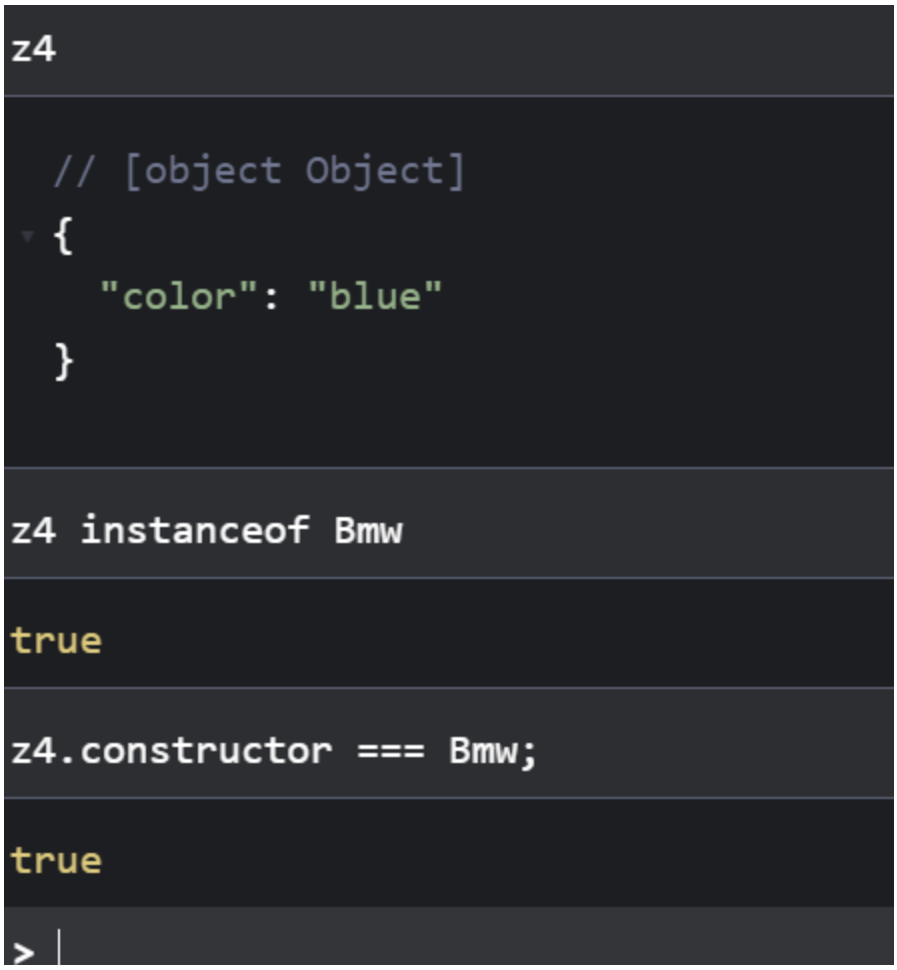



```
const Bmw = function (color) {
  this.color = color;
};

Bmw.prototype.wheels = 4;
Bmw.prototype.drive = function () {
  console.log("drive...");
};
Bmw.prototype.navigation = 1;
Bmw.prototype.stop = function(){
  console.log("STOP!");
};

const x5 = new Bmw("red");
const z4 = new BMW("blue");
```




instanceof : 생성자 함수가 새로운 객체를 만들어 낼 때 그 객체는 생성자의 인스턴스라고 한다
자바스크립트에서는 이를 편리하게 확인할 수 있는 instanceof가 있다.

객체와 생성자를 비교할 수 있고 해당 객체가 그 생성자로부터 생성된것인지 판단해서
true 혹은 false를 반환한다.



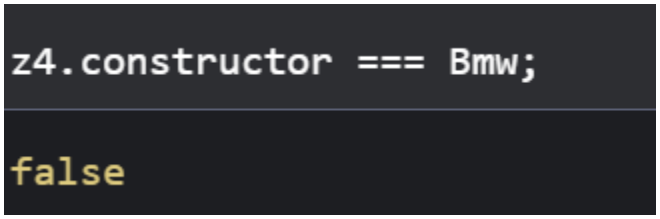

z4 는 bmw의 인스턴스다

constructor : 생성자로 만들어진 인스턴스 객체에는 constructor가 존재한다.

이 것은 생성자를 가르킨다.

```
const Bmw = function (color) {
  this.color = color;
};

Bmw.prototype.wheels = 4;
Bmw.prototype.drive = function () {
  console.log("drive...");
};
```

```
Bmw.prototype.navigation = 1;
Bmw.prototype.stop = function(){
  console.log("STOP!");
};

const x5 = new Bmw("red");
const z4 = new Bmw("blue");

이렇게도 할 수 있지만

const Bmw = function (color) {
  this.color = color;
};

Bmw.prototype = {
  wheels : 4,
  drive(){
    console.log("drive...");
  },
  navigation : 1,
  stop(){
    console.log("STOP!");
  },
};

const x5 = new Bmw("red");
const z4 = new Bmw("blue");

이렇게도 할 수 있다.
그런데 이렇게 하면 constructor가 사라진다.
```




아까는 true가 나왔었는데 지금은 false가 나온다.

이런 현상을 방지하기 위해서 프로토타입을 덮어씌우는게 아니라 하나씩 추가하는게 좋은것이다.

```
const Bmw = function (color) {
  this.color = color;
};
```

```
Bmw.prototype = {
  constructor : Bmw, // 수동으로 추가1
  wheels : 4,
  drive(){
    console.log("drive...");
  },
  navigation : 1,
  stop(){
    console.log("STOP!");
  },
};

const x5 = new Bmw("red");
const z4 = new Bmw("blue");

z4.constructor === Bmw;
true
```

수동으로 추가해주어도 된다.
자바스크립트는 명확한 constructor를 보장하지는 않는다.