EliceAcademy

JavaScript : 3주차

클래스와 프로토타입

목차

클래스가 무엇인지 알 수 있다.

클래스랑 생성자 함수 차이점을 알 수 있다.

프로토타입에 대해서 이해할 수 있다.

EliceAcademy

01

객체

값 생성 용어 정리

값(value) - 표현식이 평가되어 생성된 결과

평가(evaluate) - 값을 생성하거나 기존에 있는 값을 참조 하는 것

리터럴 (literal) - 사람이 이해할 수 있는 문자 , **약속된 기호를 사용하여 값을 생성하는 표기법**

문(statement) - 프로그램을 구성하는 기본단위, 최소 실행 단위

표현식 (expression) - '값' 으로 평가 될 수 있는 문(statement)

statement 에는 표현식인 문 , 표현식이 아닌 문 두 종류가 있다.
표현식인 문 -> 값으로 평가될 수 있는 문
표현식이 아닌 문 -> 값으로 평가될 수 없는 문

타입 변환

암묵적 타입변환

자바스크립트 엔진이 표현식을 평가할 때 개발자의 의도와는 상관없이 `코드의 문맥을 고려해 암묵적으로` 데이터 타입을 강제 변환 하는 것

`문자열(string), 숫자(number), 불리언(boolean)` 과 같은 원시 타입에서 암묵적 타입 변환이 발생

명시적 타입변환

개발자의 의도에 따라 명시적으로 타입을 변경하는 방법

1. **빌트인 생성자 함수** 사용하는 방법 → `String(), Number(), Boolean()`
2. **빌트인 메서드** 사용하는 방법 -> parseInt , .toString()
3. 암묵적 타입변환을 이용 -> 1 + `` , + '0' , !! 'x'

동등비교 , 일치비교

동등비교(loose equailty) ==

느슨한 비교 : 좌항과 우항의 피연산자를 비교할 때,
먼저 암묵적 타입 변환을 통해 타입을 일치시킨 후 , 값이 같은 지 비교

일치비교(strict equality) ===

엄격한 비교 : 좌항과 우항의 피연산자가 타입도 같고, 값도 같은지 비교,
즉, 암묵적 타입 변환을 하지 않고 값을 비교

예외) NaN은 자신과 일치하지 않는 유일한 값이다.

```
// NaN은 자신과 일치하지 않는 유일한 값
console.log(NaN === NaN); // false
isNaN(NaN); // true
isNaN(10); // false
isNaN(1 + undefined); // true
```

{ } 의 중의적 표현

{ } 은 **코드 블록** 일 수도 있고, **객체 리터럴** 일 수도 있다

{ } 이 단독으로 존재 → 자바스크립트 엔진은 { } 을 블록 스코프 으로 해석

{ } 이 값으로 평가되어야 할 문맥에서 피연산자 로 사용될 경우

→ 자바스크립트 엔진은 { }을 리터럴 객체로 해석

원시타입 과 객체타입 차이점

원시 타입(primitive type)

단 한가지의 값을 저장하는 타입 , **변경 불가한 값**

순수 값의 주소를 참조

변수의 복사나 참조 => 새로운 메모리방을 만든다 : 깊은 복사

객체 타입(reference type)

여러 개의 데이터가 저장되는 타입, **변경 가능한 값 (할당 된 값 수정 가능)**

동적메모리에 저장된 주소 값을 참조

변수의 복사나 참조 => 이미 만들어진 기존 주소 값을 복사 : 얕은 복사

[ ] , { } , 생성자 함수 , new 를 이용하면 새로운 메모리 방이 만들어진다

유사 배열 객체 : String 문자열

원시 값은 변경 불가능한 값(immutable value)
원시 타입 변수의 값을 재할당하면 새로운 메모리에 새로운 공간을 만든다

유사 배열 객체 => 읽기 전용 배열 , 수정 불가능
배열처럼 인덱스로 프로퍼티 값에 접근할 수 있으며, length 프로퍼티를 갖는 객체
length 프로퍼티와 인덱스를 가지고 있어서 **for 문으로 순회도 가능**

```
let str = '박연미';
console.log(str[0], str.length); // 박 3
str[0] = '김';
console.log(str); // 박연미

str = '박연미';
str = '김연미'; // 재할당
console.log(str); // 김연미
```

```
let strArr = ['박', '연', '미'];
strArr[0] = '김';
console.log(strArr); //[ '김', '연', '미' ]
```

객체 , 프로퍼티 , 메서드

참고:
https://developer.mozilla.org/ko/docs/Web/JavaScript/Guide/Working_with_Objects

객체(Object)

물체를 프로그래밍으로 구현한 것으로 **물체의 정보**와 **물체가 하는 행동**으로 구성된다

객체 내부에는 프로퍼티(정보) 와 메서드(행동) 가 **객체 멤버**로 들어 있다

프로퍼티(property)

객체 안에 들어있는 속성 값으로 (key : value) 그 객체에 대한 상태 정보를 표현

메서드(method)

객체 안에 들어있는 함수로 객체지향에서는 **객체 안의 함수**를 메서드라고 부름

프로퍼티(상태 데이터)를 참조하고 조작할 수 있는 동작(기능, 행동)을 담긴 함수로 표현

프로퍼티

프로퍼티(property : 속성) => key : value

key : String, (Number, Symbol 는 암묵적으로 문자타입으로 변환) : **식별자로 역할**

value : 모든 타입이 들어올 수 있다 => 참조 변수는 주소 값이 들어있다

이미 존재하는 key값을 여러 번 선언하면 값을 덮어쓰기

없는 key의 값을 참조하면 undefined

1. 마침표 표기법 → 정적인 이름을 .(점)으로 key 값 접근
2. 대괄호 표기법 ->[ ] key 에 특수문자를 쓴 경우
   -> **동적으로 접근할 경우 => [변수이름 , 식별자 사용가능]**

delete 키워드 사용해서 프로퍼티 삭제

프로퍼티

프로퍼티(property : 속성) => key : value

key : String, (Number, Symbol 는 암묵적으로 문자타입으로 변환)

value : 모든 타입이 들어올 수 있다 => 참조 변수는 주소 값이 들어있다

이미 존재하는 key값을 여러 번 선언하면 값을 덮어쓰기

없은 key의 값을 참조하면 undefined

delete 키워드 사용해서 속성값 삭제

this

this는 **자신이 속한 객체 또는 자신이 생성할 인스턴스를 가리키는 자기 참조 변수**(self-reference variable)이다. this를 통해 자신이 속한 객체 또는 자신이 생성할 인스턴스의 프로퍼티나 메서드를 참조할 수 있다

브라우저 호스트 환경의 JS의 모든 변수, 함수는 window라는 객체의 프로퍼티와 메소드
Node.js 호스트 환경의 JS의 모든 변수, 함수는 global라는 객체의 프로퍼티와 메소드

this(객체 주소 변수)는 자바스크립트 엔진에 의해 암묵적으로 생성
this는 코드 어디서든 참조 가능 하지만 this 값이 어디서 호출 하느냐 따라 변동되는 값!
일반적으로 객체의 메서드 내부 또는 생성자 함수 내부에서만 this 사용
함수를 호출하면 자신이 속한 객체의 this가 암묵적으로 함수 내부에 전달

02

클래스

EliceAcademy

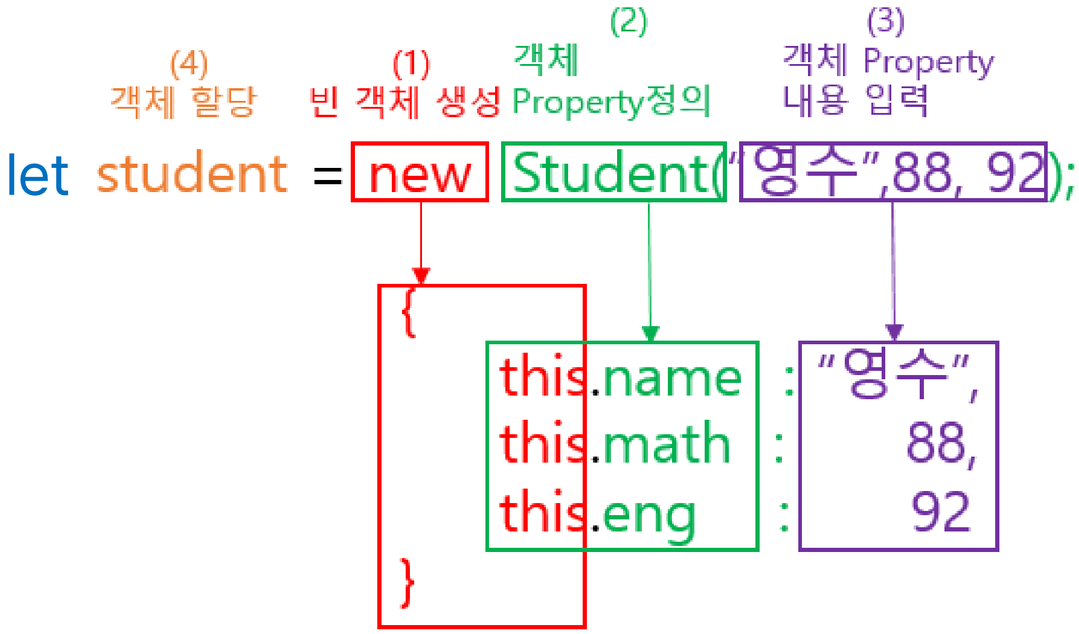
생성자 함수란?

반복적으로 재사용가능한 객체를
생성할 때 사용하는 함수

ES6에서 클래스 개념을 사용하기 이전
생성자 함수로 재사용한 객체

생성자 함수 이름은 대문자로 한다
new 라는 키워드를 통해 객체를 생성한다

(1) new 키워드는 빈 객체 {}를 생성
(2) 생성자 함수는 빈 객체에 생성 할 프로퍼티를 정의
(3) 함수의 인수들을 프로퍼티에 할당
(4) 생성된 Student 객체를 student에 할당



* This는 생선 된 객체 차제를 의미합니다.

클래스

객체를 생성하기 위한 템플릿으로 ES6 부터 추가된 문법
데이터(속성)과 행동(메서드)을 하나의 **객체로 추상화** 한다

클래스 사용하는 방법

```
class 키워드 클래스 이름 (대문자 영어 명사단어)
{
  constructor(){ } // new 할 때 호출되는 생성자 함수 부분
                   // 보통 내부에 속성 값의 초기값을 정의한다

  method1(){ } // this 나 function 키워드 없이 함수이름(){} 으로 선언

}
```

참고:
https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Statements/class

EliceAcademy

static 스테틱

클래스 변수
(static fields)

"**공유**"를 위한 객체 외부 변수

VS

인스턴스 변수
(non-static fields)

객체 내부에 존재하는 변수

클래스 메소드
(static methods)

"**클래스**"가 동작시키는 메소드
=> 간편 기능 구현에 유용
=> Math.random();

VS

인스턴스 메소드
(non-static methods)

"**주 객체**"가 동작하는 메소드
=> 특정 객체의 동작 구현 시 유리
=> ma.stimpack();

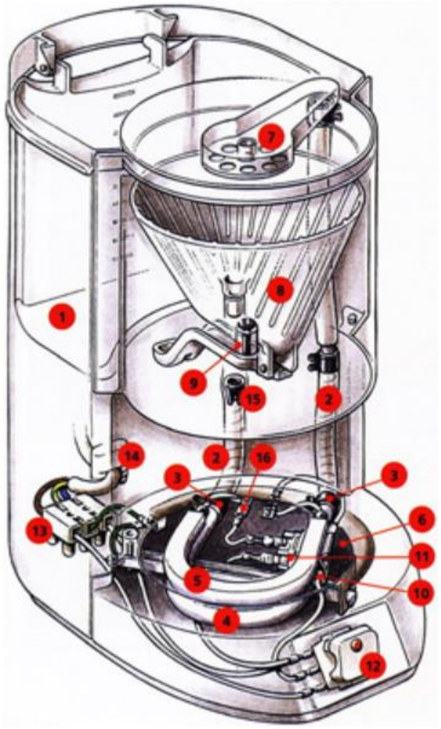
참고:
https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Classes/static

https://cloudstudying.kr/lectures/198

private 접근 제어자

public :
클래스 밖에서도 접근 가능한 프로퍼티와 메서드
기본적으로 아무것도 안 붙어있으면 다 public 이다

private :
오직 클래스 내부에서만 접근할 수 있다.
외부에서 출력해도 보이지 않는다
변수 이름 앞에 #을 붙이면 된다



캡슐화(encapsulation)

객체의 필드(속성), 메소드를 하나로 묶고,
실제 구현 내용을 외부에 감추는 것

외부 객체는 객체 내부의 구조를 얻지 못하며
객체가 노출해서 제공하는 필드와 메소드만 이용 가능

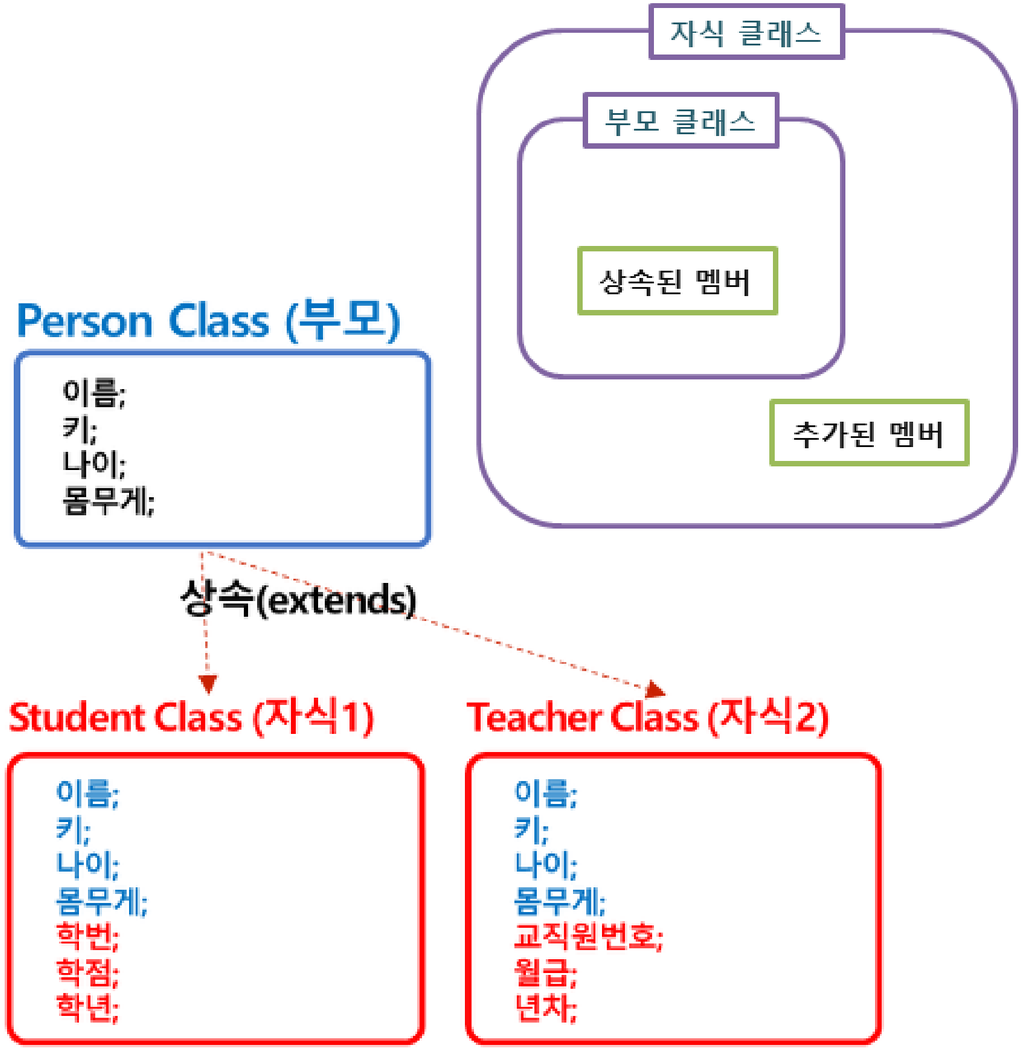
필드와 메소드를 캡슐화 하여 보호하는 이유는
외부의 개입 방지위해서, 데이터 보호차원이다.

접근자 프로퍼티 getter setter 이용하여 간접적으로
필드와 메서드 제공 해줌

EliceAcademy

상속 (inheritance)

기존의 클래스에 기능을 추가하거나 재정의하여 새로운 클래스를 정의하는 것을 의미
부모 클래스의 모든 필드와 메소드를 물려받아, 자식 클래스를 생성



상속의 장점

1. 클래스 계층적 구조화 가능 => 유지보수 쉬움
2. 클래스 안에 요소 중복 최소화

super() => 부모 생성자 호출

03

프로토타입

프로토타입이란?

자바스크립트의 프로토 타입을 기반으로 상속을 구현한다
부모 객체의 프로퍼티 또는 메소드를 상속받아 사용할 수 있게 한다.
이 부모 객체를 Prototype(프로토타입) 객체 혹은 그냥 Prototype(프로토타입) 이라고 한다
최상위 Prototype 객체는 Object

모든 객체는 자신의 프로토타입 객체를 가리키는
[[Prototype]] 인터널 슬롯(internal slot) 을 갖으며 상속을 위해 사용된다.

함수도 객체이므로 [[Prototype]] 인터널 슬롯을 갖는다.
그런데 함수 객체는 일반 객체와는 달리 prototype 프로퍼티가 추가로 있다

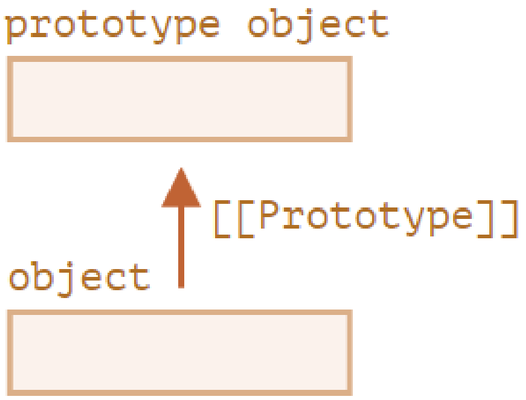
참고:
https://poiemaweb.com/js-prototype
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object

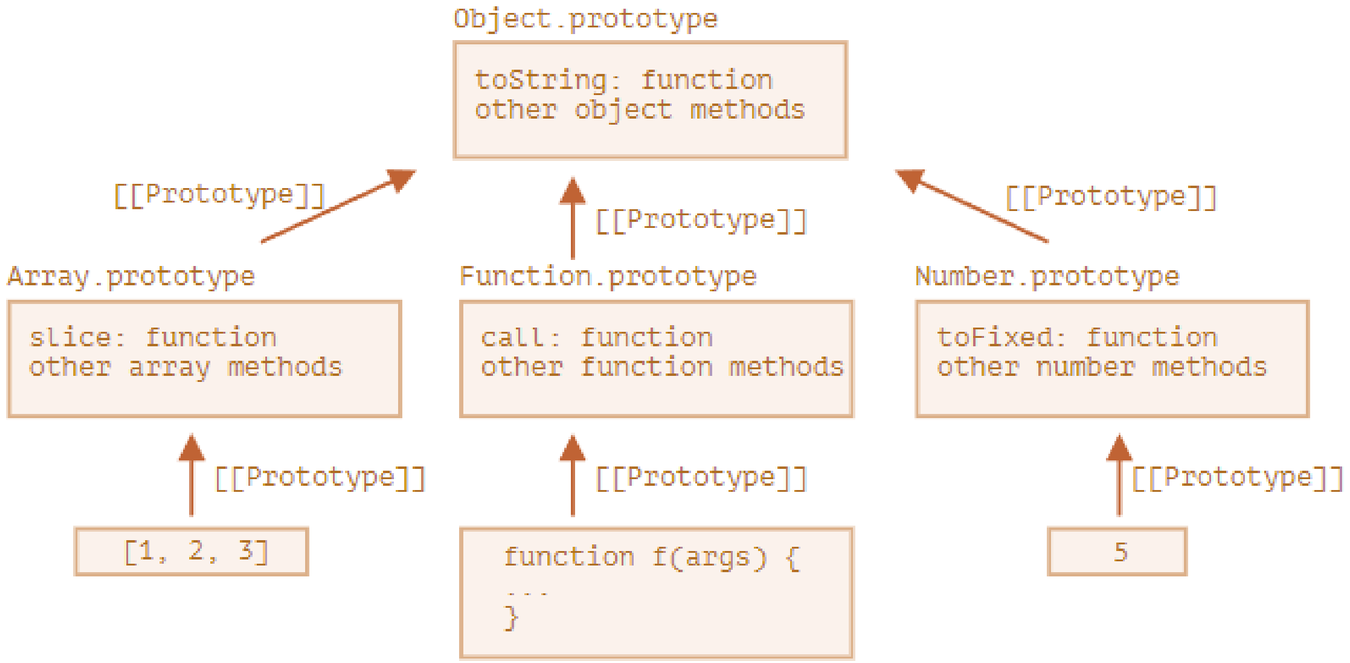
EliceAcademy

프로토타입과 프로토타입 체인

객체간 상속의 연결고리는
프로토타입 체인으로 연결 되어 있다

클래스도 프로토타입 체인구조 이다
문법적으로 깔끔하게 표현한 것





EliceAcademy

prototype 과 _ _proto_ _

```
const obj1 = {};
const obj2 = new Object();
const obj3 = new Person();
```

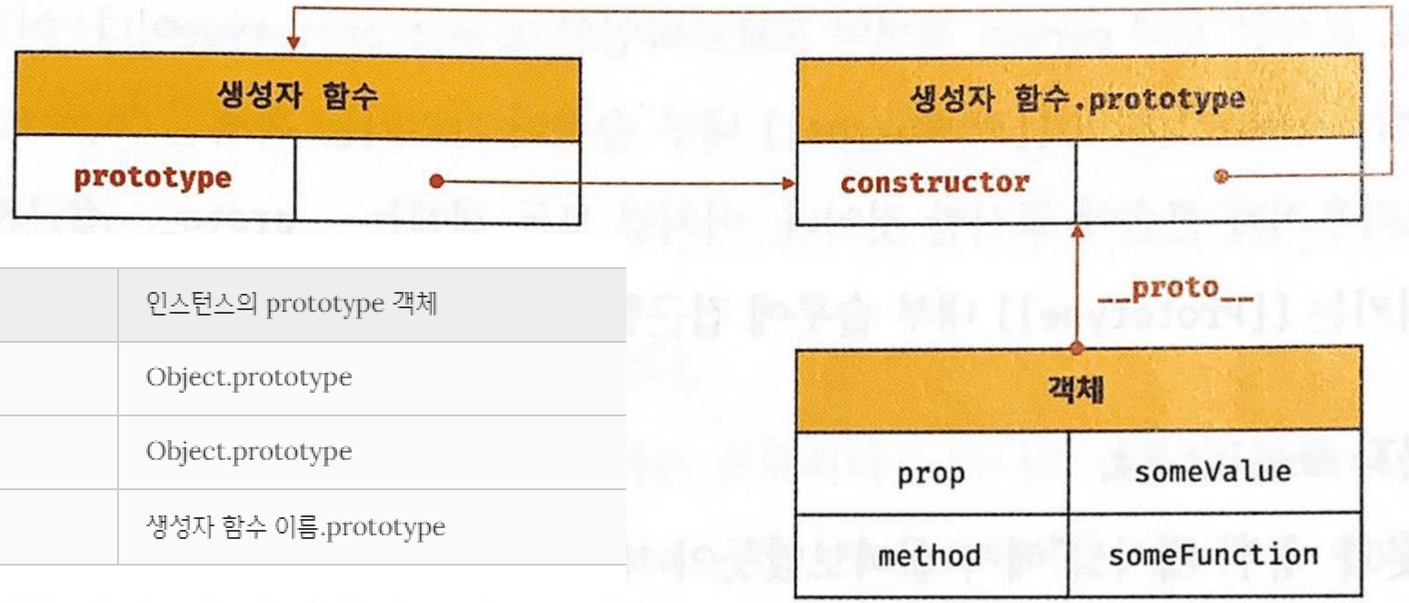


그림 19-3 객체와 프로토타입과 생성자 함수는 서로 연결되어 있다.

객체 생성 방식	엔진의 객체 생성	인스턴스의 prototype 객체
객체 리터럴	Object() 생성자 함수	Object.prototype
Object() 생성자 함수	Object() 생성자 함수	Object.prototype
생성자 함수	생성자 함수	생성자 함수 이름.prototype

property 는 함수 객체만 가지고 있고 이는 함수 객체가 생성자로 사용될 때 이 함수를 통해 생성될 객체의 부모 역할을 하는 객체(프로토타입 객체)를 가리킨다.

프로퍼티 어트리뷰트 : 데이터 프로퍼티

프로퍼티 어트리뷰트	프로퍼티 디스크립터 객체의 프로퍼티	설명
[[Value]]	Value	■ 프로퍼티 키를 통해 프로퍼티 값에 접근하면 반환되는 값이다. ■ 프로퍼티 키를 통해 프로퍼티 값을 변경하면 [[Value]]에 값을 재할당한다. 이때 프로퍼티가 없으면 프로퍼티를 동적으로 생성하고 생성된 프로퍼티의 [[Value]]에 값을 저장한다.
[[Writable]]	Writable	■ 프로퍼티 값의 변경 가능 여부를 나타내며 불리언 값을 갖는다. ■ [[Writable]]의 값이 false인 경우 해당 프로퍼티의 [[Value]]의 값을 변경할 수 없는 읽기 전용 프로퍼티가 된다.
[[Enumerable]]	Enumerable	■ 프로퍼티의 열거 가능 여부를 나타내며 불리언 값을 갖는다. ■ [[Enumerable]]의 값이 false인 경우 해당 프로퍼티는 for…in 문이나 Object.keys 메서드 등으로 열거할 수 없다.
[[Configurable]]	Configurable	■ 프로퍼티의 재정의 가능 여부를 나타내며 불리언 값을 갖는다. ■ [[Configurable]]의 값이 false인 경우 해당 프로퍼티의 삭제, 프로퍼티 어트리뷰트 값의 변경이 금지된다. 단, [[Writable]]이 true인 경우 [[Value]]의 변경과 [[Writable]]을 false로 변경하는 것은 허용된다

프로퍼티 어트리뷰트 : 접근자 프로퍼티

프로퍼티 어트리뷰트	프로퍼티 디스크립터 객체의 프로퍼티	설명
[[Get]]	get	■ 접근자 프로퍼티를 통해 데이터 프로퍼티의 값을 읽을 때 호출되는 접근자 함수이다. 즉, 접근자 프로퍼티 키로 프로퍼티 값에 접근하면 프로퍼티 어트리뷰트 [[Get]]의 값, 즉 getter 함수가 호출되고 그 결과가 프로퍼티 값으로 반환된다.
[[Set]]	set	■ 접근자 프로퍼티를 통해 데이터 프로퍼티의 값을 저장할 때 호출되는 접근자 함수이다. 즉, 접근자 프로퍼티 키로 프로퍼티 값을 저장하면 프로퍼티 어트리뷰트 [[Set]]의 값, 즉 setter 함수가 호출되고 그 결과가 프로퍼티 값으로 저장된다.
[[Enumerable]]	enumerable	■ 데이터 프로퍼티의 [[Enumerable]] 과 같다.
[[Configurable]]	configurable	■ 데이터 프로퍼티의 [[Configurable]] 과 같다.