3강

프로토타입, 상속과 클래스

WHATEVER YOU WANT, MAKE IT REAL.

강사 정길용

LIKE LION

{ 프로토타입

상속

클래스 }

일급 객체(First-class object)

- 변수, 배열 엘리먼트, 다른 객체의 프로퍼티에 할당될 수 있다.
- 함수의 인자로 전달될 수 있다.
- 함수의 결과 값으로 반환될 수 있다.
- 리터럴로 생성될 수 있다.
- 동적으로 생성된 프로퍼티를 가질 수 있다.

자바스크립트의 함수(Function)는 일급 객체이다.

- 함수 == (호출 + 객체)

함수가 일급 객체라서 가능한 일

- 콜백 함수(Callback function)
  - 다른 함수에 인자로 전달되어 어떤 작업의 결과로 호출되는 함수
- 고차 함수(Higher order function)
  - 함수를 인자로 받거나 반환하는 함수
- 클로저(Closure)

LIKE LION

ex03-01.js 

prototype

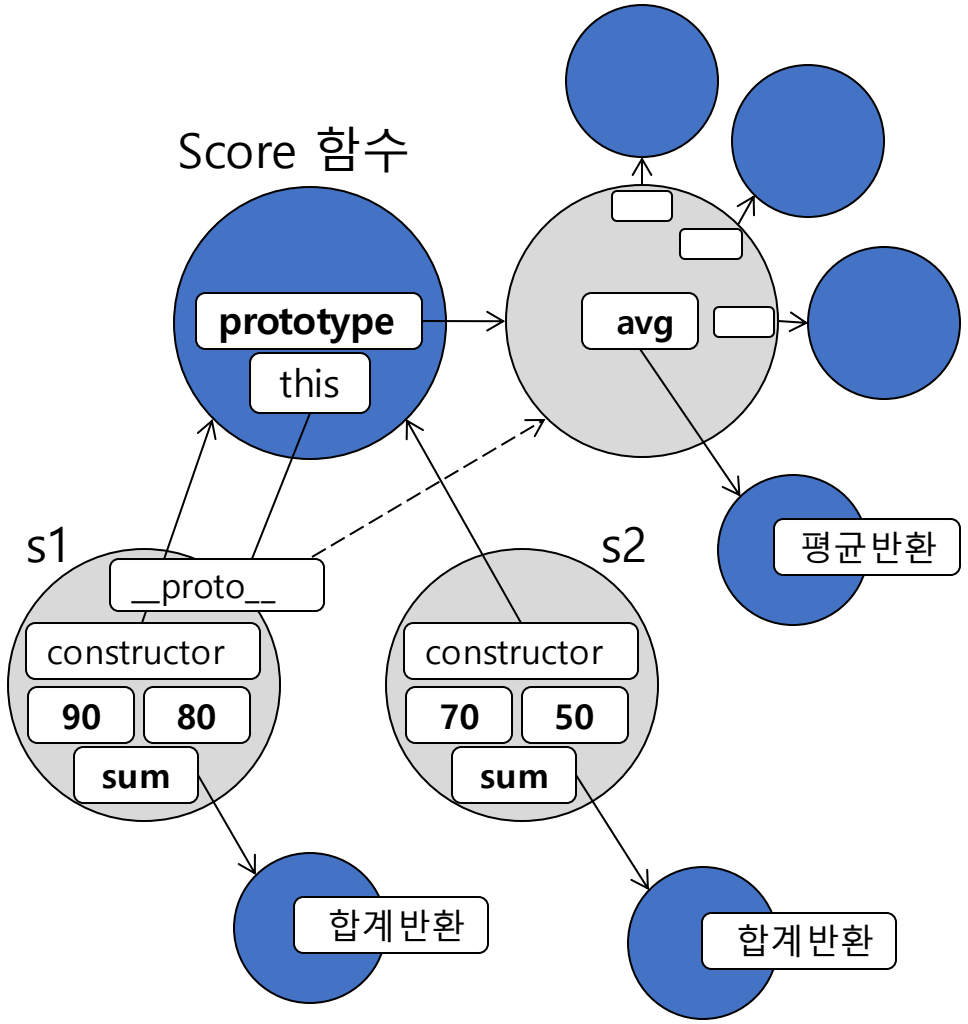
- 모든 함수가 기본으로 가지고 있는 속성
- 초기값은 빈 객체이다.
- prototype에 추가한 속성은 해당 함수가 생성자로 사용될 때 생성된 인스턴스에서 내부 링크로 참조되어 사용된다.
- 결국, prototype은 생성자 함수를 통해 생성되는 인스턴스의 메서드를 정의하는 역할을 한다.

생성자 내부에 메서드 정의 vs. prototype에 메서드 정의

```
function Score(kor, eng){
  this.kor = kor;
  this.eng = eng;
  this.sum = function(){
    return this.kor + this.eng;
  };
}
Score.prototype.avg = function(){
  return this.sum() / 2;
};
```

▶ prototype과 주변 객체의 참조관계

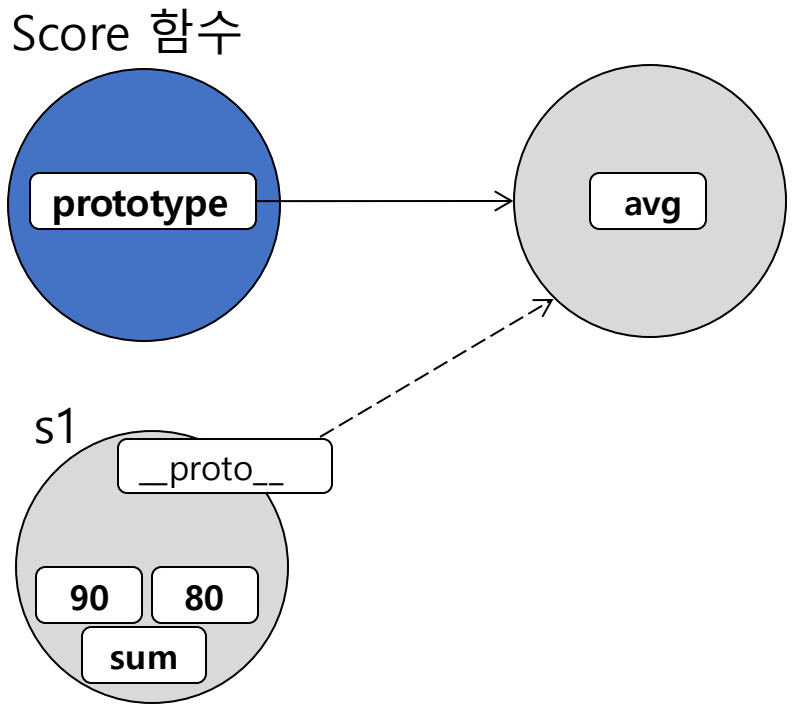
```
function Score(kor, eng){
  this.kor = kor;
  this.eng = eng;
  this.sum = function(){
      return this.kor+this.eng;
  };
}

Score.prototype.avg = function(){
  return this.sum()/2;
};

const s1 = new Score(90, 80);
const s2 = new Score(70, 50);
s1.sum();
s1.avg();
s2.sum();
s2.avg();
```



▶ 객체 초기화 순서

1. 생성자 함수의 prototype 속성의 객체가 새로 만들어진 객체 인스턴스와 바인딩된다.
   - __proto__ 속성이나 Object.getPrototypeOf() 메서드로 접근 가능

2. 생성자 함수 내에서 this에 정의한 속성들이 객체 인스턴스에 추가된다.



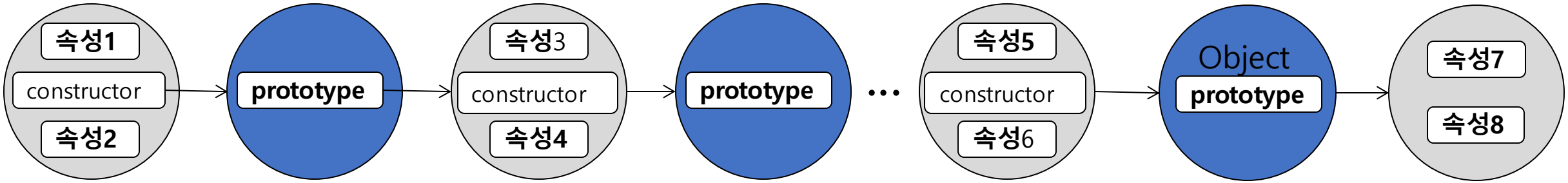
▶ 객체의 프로퍼티 참조 순서(프로토타입 체인)

1. 객체에 해당 프로퍼티가 있으면 사용한다.
2. 객체에 연결된 프로토타입에 해당 프로퍼티가 있으면 사용한다.
3. 프로토타입에도 해당 프로퍼티가 없으면 연결된 프로토타입에서 찾는다.(찾을때까지 반복)
4. 최상위 프로토타입인 Object까지 찾아봐서 해당 프로퍼티가 없다면 그 값은 undefined가 된다.

즉, 프로퍼티 참조는 해당 객체에서 먼저 찾고, 실패했을 때 프로토타입을 확인함

▶ constructor

- 모든 객체에 정의되어 있는 속성
- 해당 객체를 만드는데 사용된 생성자를 참조



LIKE LION

ex03-01.js 

▶ 프로토타입 체인의 마지막 객체

- 모든 객체의 prototype 체인 마지막 객체는 Object이다.
- 즉, Array, String, Number, RegExp, Date, Function 등의 내장 객체와 Score, Ping 등 사용자가 정의한 객체는 모두 프로토타입 체인에 의해서 자동으로 Object의 메서드를 사용할 수 있다.

▶ 내장된 생성자 함수의 prototype

- Object, Array, Function, Date, XMLHttpRequest 등
- 내장된 생성자 함수도 prototype 속성이 있으므로 이곳에 속성, 메서드를 추가해서 네이티브 객체의 기능을 확장할 수 있다.

▶ typeof 연산자

- 객체의 타입을 반환
  - typeof "hello" -> "string"
  - typeof 10 -> "number"
  - typeof true -> "boolean"
  - typeof function(){} -> "function"
  - typeof [] -> "object"
  - typeof {} -> "object"
  - typeof new Score() -> "object"

- 기본 데이터 타입과 함수를 제외한 모든 인스턴스에 대해 object 반환

LIKE LION

ex03-02.js, 03-03

instanceof 연산자

- 객체가 지정한 생성자를 통해서 생성되었는지 판단
- 직접 생성된 생성자가 아니더라도 프로토타입 체인에 있는 생성자라면 true 반환
    - `new Score() instanceof Object -> true`
- JSON 표기법으로 생성한 배열이나 객체는 내부적으로 Array, Object 생성자 함수를 통해 생성이 된다.

```
new Array() instanceof Array          -> true
[] instanceof Array                   -> true
new Object() instanceof Object        -> true
({}) instanceof Object                -> true

new Score() instanceof Score          -> true
new Score() instanceof Object         -> true
```

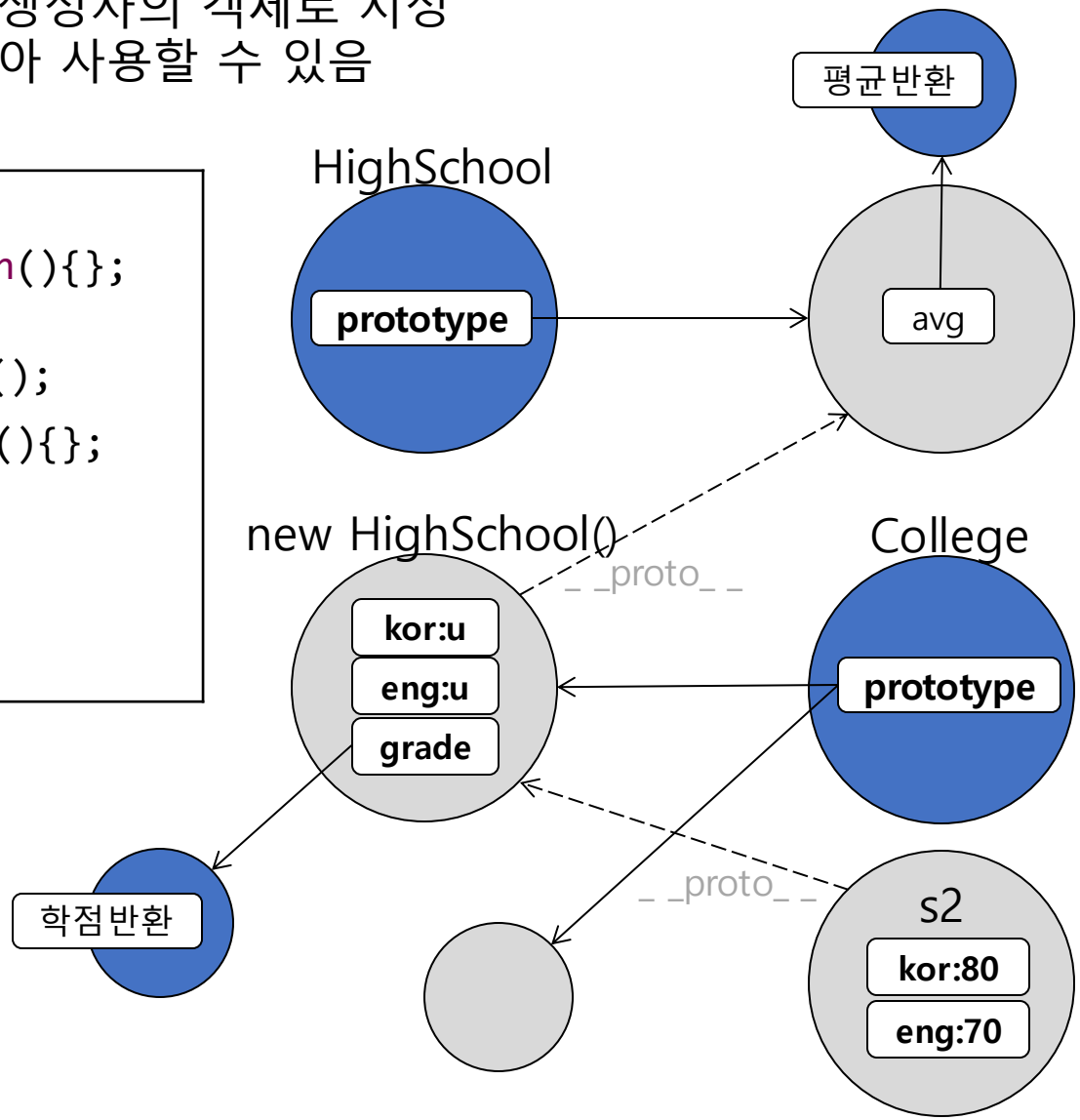
LIKE LION

ex03-04.js

프로토타입 체인을 이용한 상속 기능 구현

- 하위 생성자의 프로토타입을 상위 생성자의 객체로 지정
- 상위 생성자의 모든 속성을 물려받아 사용할 수 있음

```
function HighSchool(){...}
HighSchool.prototype.avg = function(){};
function College(){...}
College.prototype = new HighSchool();
College.prototype.grade = function(){};
const s2 = new College(80, 70);
s2.avg();
s2.grade();
```

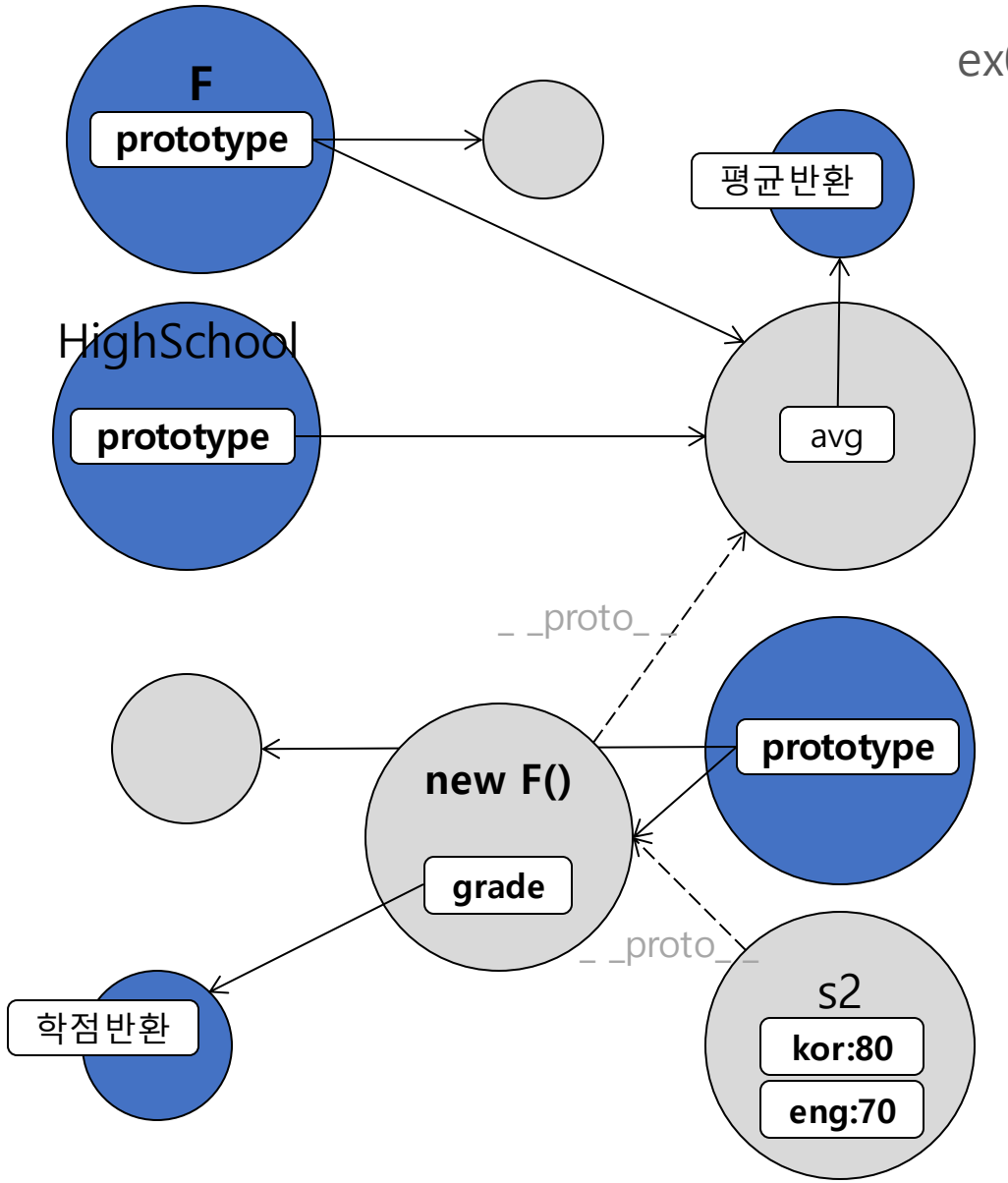


LIKE LION

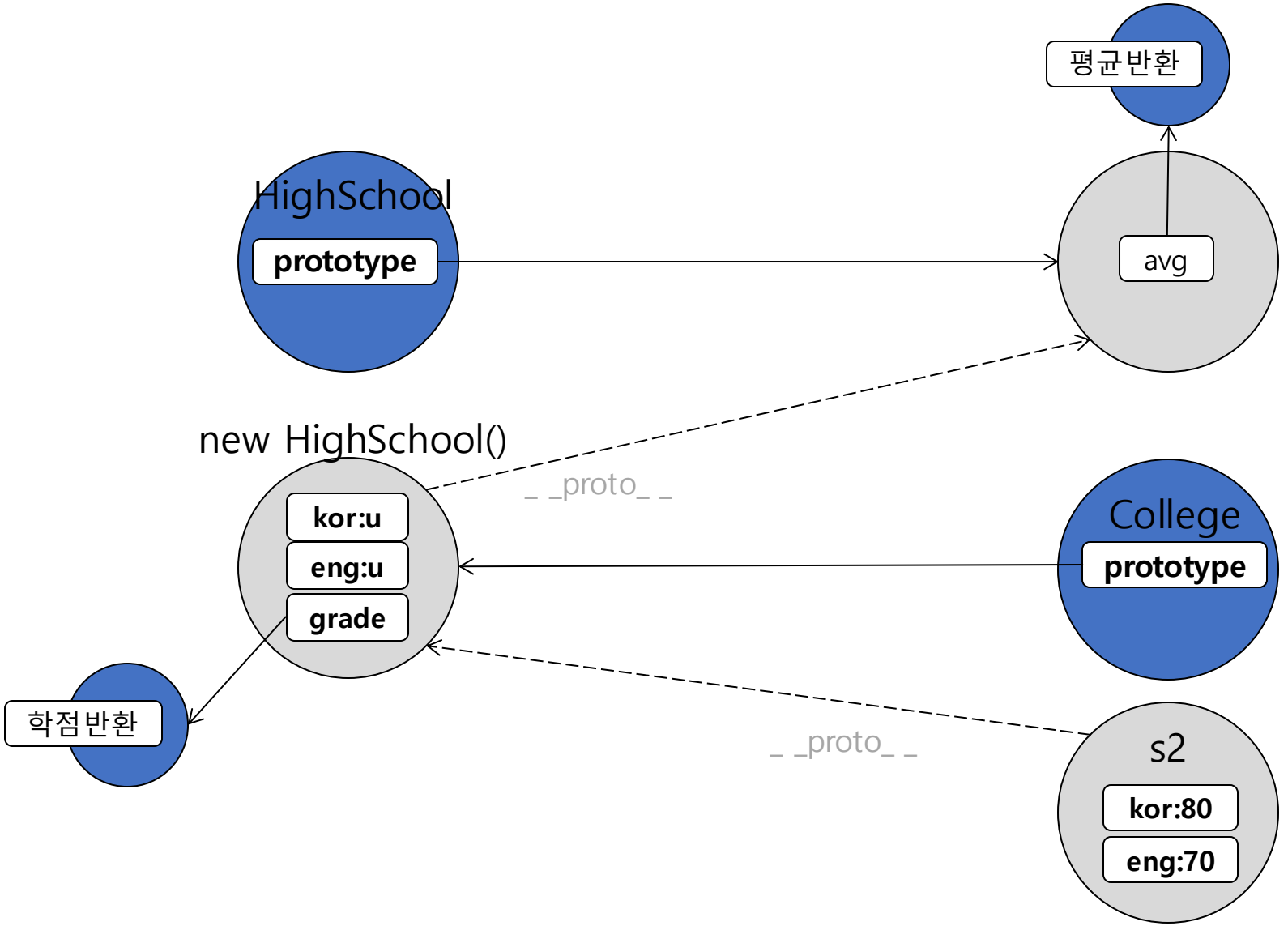
ex03-05.js

```
function HighSchool(){...}
HighSchool.prototype.avg = function(){};
function College(){...}
inherite(HighSchool, College);
College.prototype.grade = function(){};
const s2 = new College(80, 70);
s2.avg();
s2.grade();
```

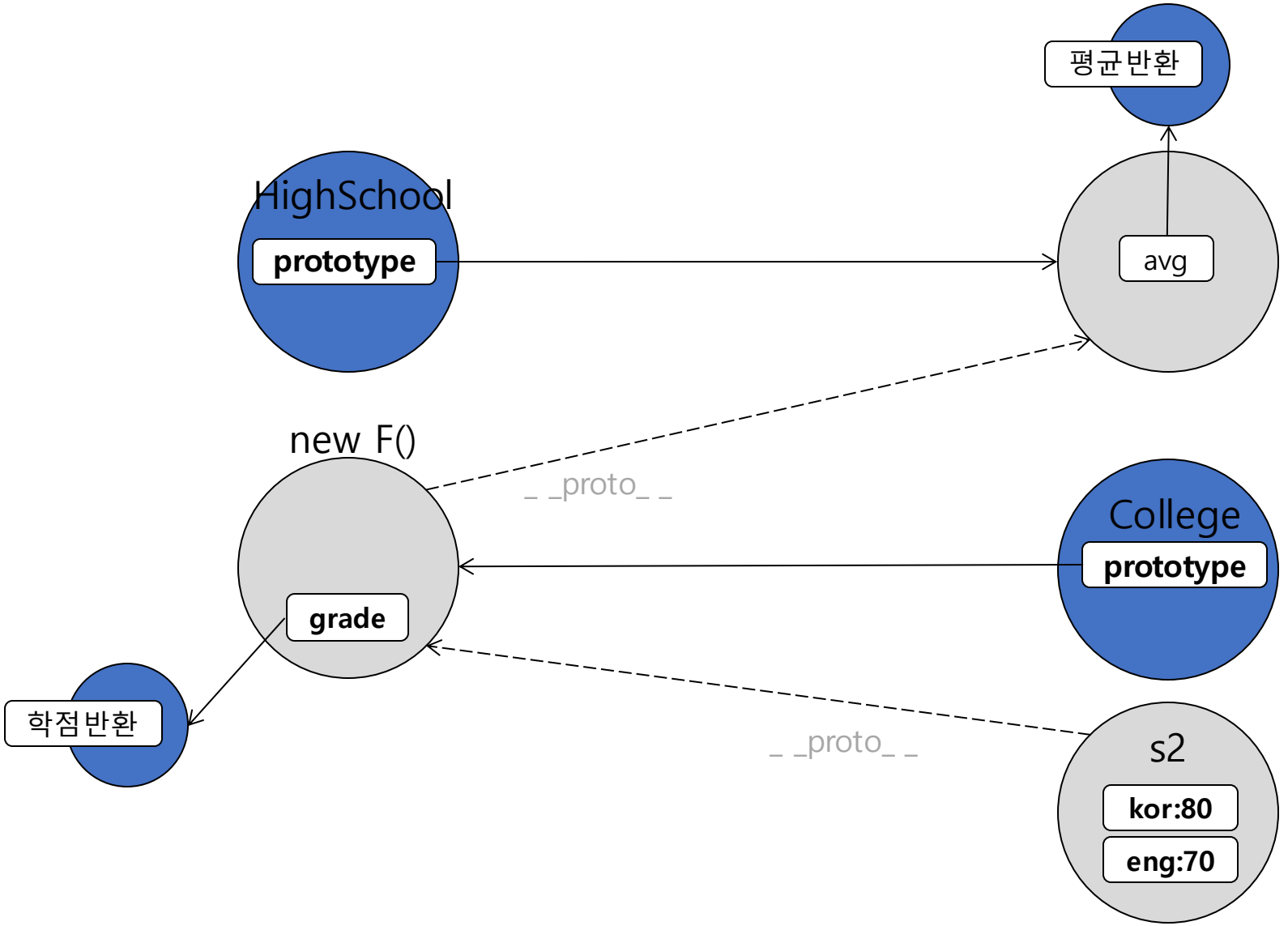
```
function inherite(Parent, Child){
    const F = function(){};
    F.prototype = Parent.prototype;
    Child.prototype = new F();
}
```



LIKE LION


평균반환
HighSchool
prototype
avg
new HighSchool()
_ _proto_ _
kor:u
eng:u
grade
College
prototype
학점반환
_ _proto_ _
s2
kor:80
eng:70

LIKE LION


평균반환
HighSchool
prototype
avg
new F()
_ _proto_ _
College
prototype
grade
학점반환
_ _proto_ _
s2
kor:80
eng:70

class

- ECMAScript6(2015)에 추가된 키워드
- 객체지향 언어의 class와 비슷한 방식으로 생성자 함수를 기술
- 객체를 생성하고 prototype 기반의 상속을 보다 명료하게 표현
- class는 사실 함수이며 class 선언문과 class 표현식 방식으로 사용

class 선언문

- class 클래스명{ …… }

```
class HighSchool{

}
```

class 표현식

- const SomeClass = class [클래스명]{ …… }

```
const HighSchool = class {

};
```

Class body와 메서드 정의

- 클래스의 바디에 클래스 멤버변수(속성)와 메서드 정의
- 멤버변수의 초기화는 Constructor 메서드에 정의

Constructor 메서드(생성자)

- 객체를 생성하고 초기화 하는 메서드(생략 가능)
- 주로 클래스 멤버변수를 초기화하는 작업
- constructor라는 이름으로 작성
- 하나만 작성 가능
- super()로 부모의 생성자 호출 가능

```
class HighSchool{
  constructor(kor, eng){
    this.kor = kor;
    this.eng = eng;
  }
}
const s1 = new HighSchool(100, 90);
console.log(s1.kor, s1.eng);
```

```
function HighSchool(kor, eng){
  this.kor = kor;
  this.eng = eng;
}
const s1 = new HighSchool(100, 90);
console.log(s1.kor, s1.eng);
```

Prototype 메서드 정의

- 클래스의 prototype에 지정할 메서드 정의
- 클래스의 인스턴스를 생성한 후 인스턴스.메서드명() 으로 호출 가능
- 메서드명(){ …… }

```
class HighSchool{
  constructor(kor, eng){
    this.kor = kor;
    this.eng = eng;
  }
  sum(){
    return this.kor + this.eng;
  }
  avg(){
    return this.sum() / 2;
  }
}
const s1 = new HighSchool(100, 90);
console.log(s1.sum());
```

```
function HighSchool(kor, eng){
  this.kor = kor;
  this.eng = eng;
}

HighSchool.prototype.sum = function(){
  return this.kor + this.eng;
};
HighSchool.prototype.avg = function(){
  return this.sum() / 2;
};

const s1 = new HighSchool(100, 90);
console.log(s1.sum());
```

LIKE LION

ex03-06.js 

▶ extends

- 상속을 통해 자식 클래스를 정의
- class 자식클래스명 extends 부모클래스명 { ..... }

```
class College extends HighSchool{
  constructor(kor, eng){
    // 부모의 constructor 호출
    super(kor, eng);
  }
  // 부모의 avg 메서드 재정의
  avg(){
    ......
  }
  // 새로운 메서드 추가
  grade(){
    ......
  }
}
const s2 = new College(80, 70);
console.log(s2.sum());
console.log(s2.grade());
```

```
function College(kor, eng){
  HighSchool.call(this, kor, eng);
}
inherite(HighSchool, College);
function inherite(Parent, Child){
  Child.prototype = Object.create(Parent.prototype);
  Child.prototype.constructor = Child;
};
College.prototype.avg = function(){
  ......
};
College.prototype.grade = function(){
  ......
};
const s2 = new College(80, 70);
console.log(s2.sum());
console.log(s2.grade());
```