

# Prototype Chaining

# 목차

1. 프로토타입

2. 프로토타입 체이닝

## 1. 프로토타입

자바스크립트는 프로토타입 기반 언어이다

자바스크립트의 객체는 프로토타입을 갖는다

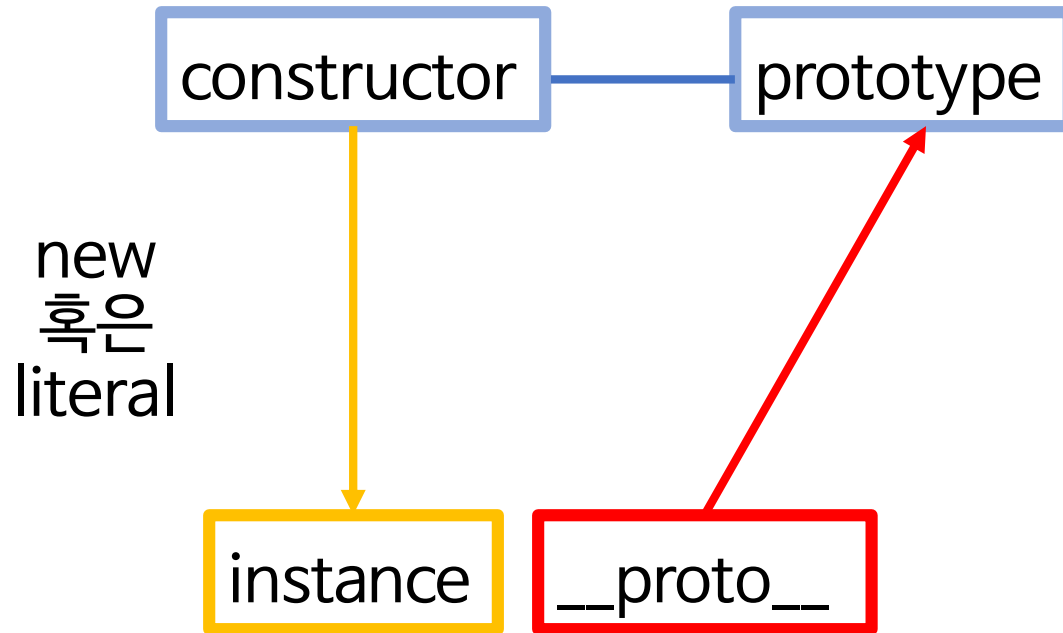
prototype, constructor, \_\_proto\_\_

## 1. 프로토타입



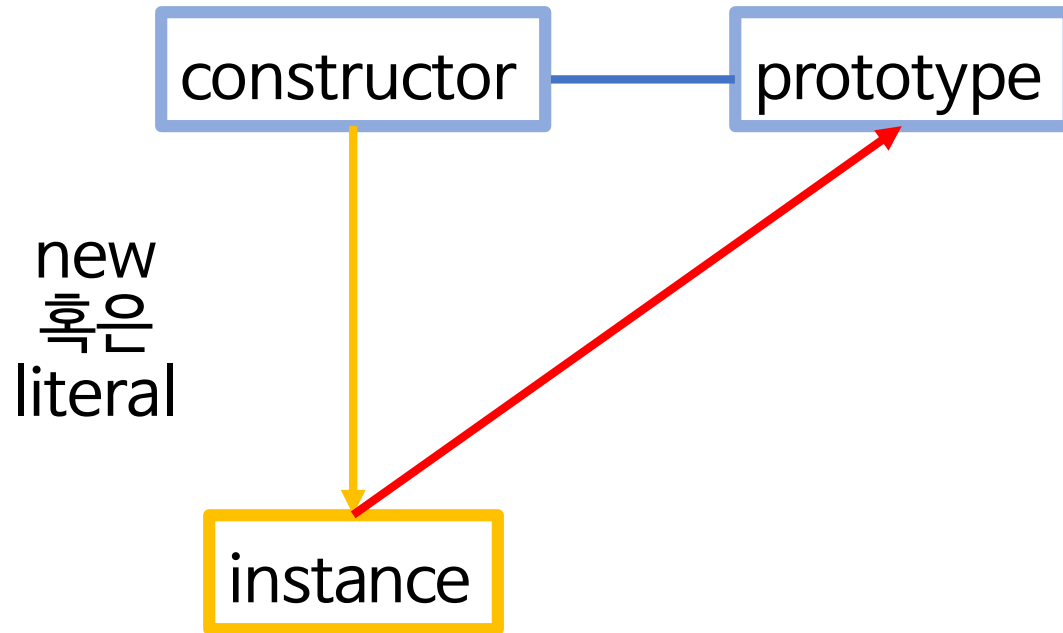
constructor라는 생성자 함수는 prototype을 갖는다

## 1. 프로토타입



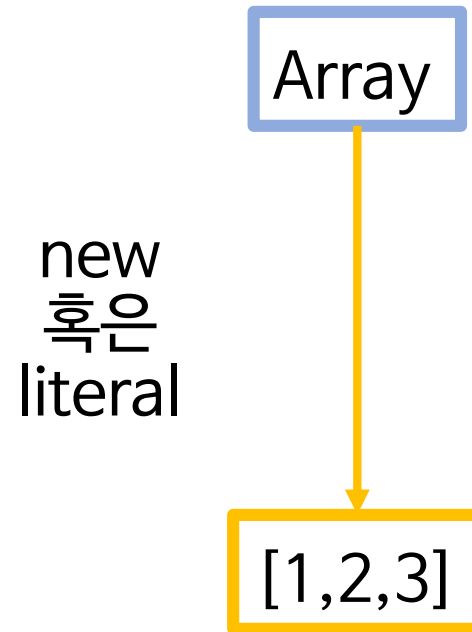
New 혹은 리터럴 (`{}`, `[]`)을 통해 인스턴스를 생성하면  
인스턴스의 `__proto__`는 `prototype`을 가리키게 된다.  
즉 생성자의 `prototype`과 인스턴스의 `__proto__`는 같다.

## 1. 프로토타입



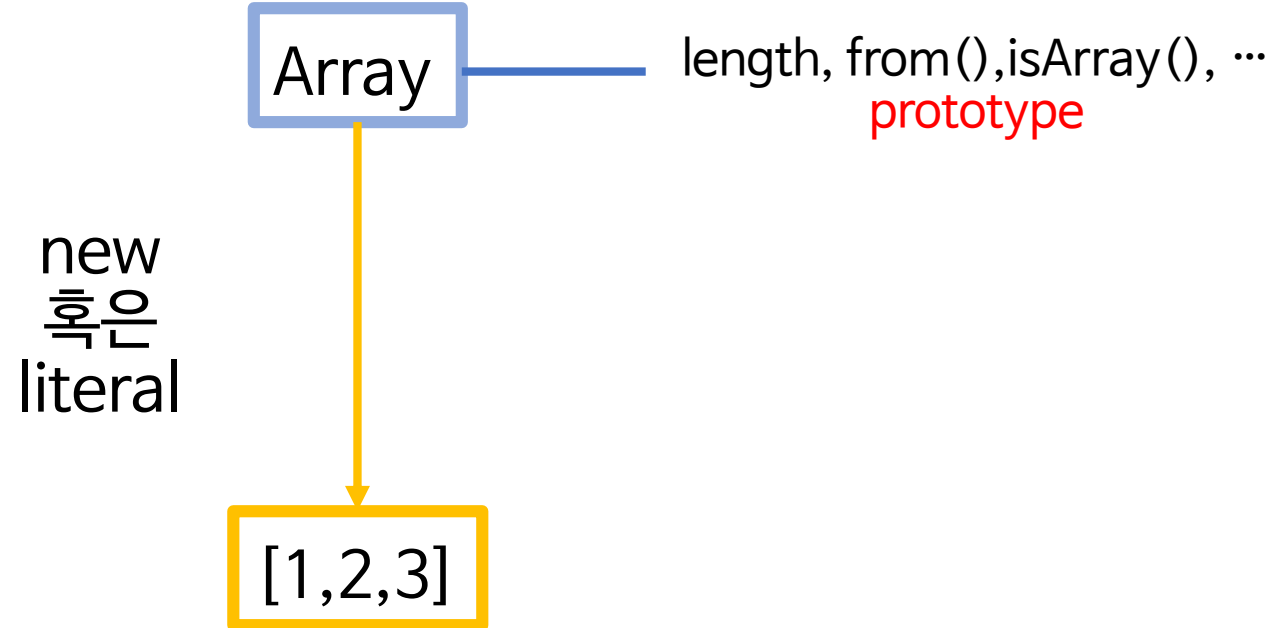
Instance에서 prototype을 쓸 때, `__proto__`는 생략가능하다.  
(실제로는 아니지만 그런것처럼 작동한다)

## 1. 프로토타입



Array생성자를 통해 배열을 생성한다.

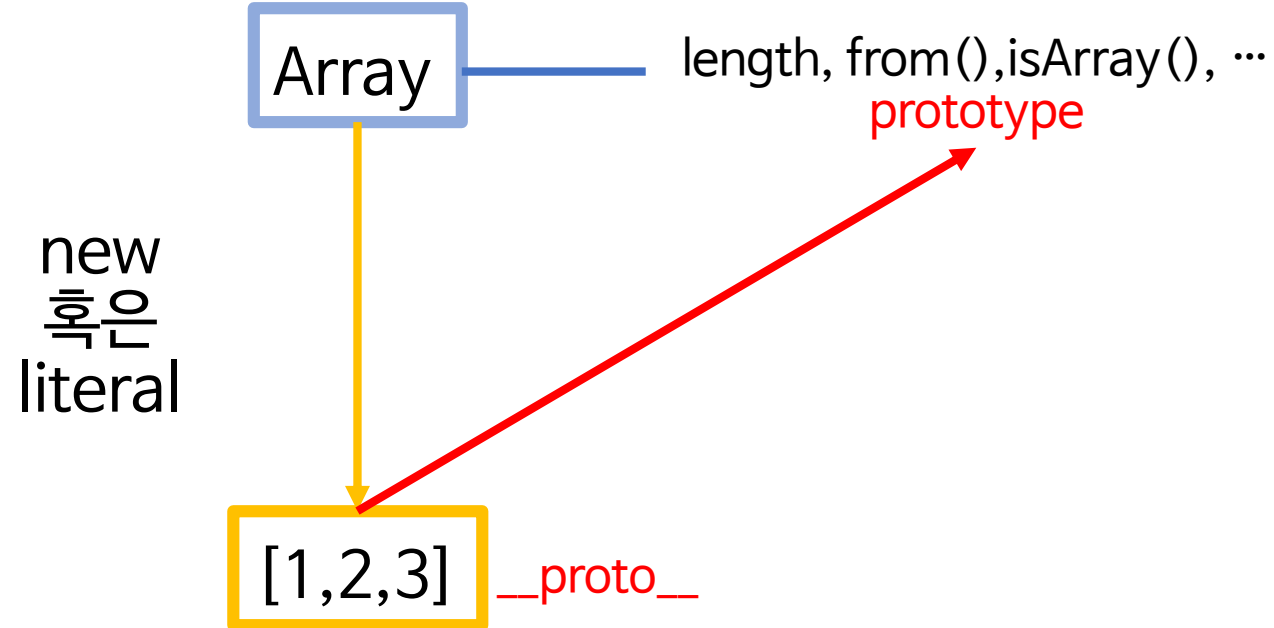
# 1. 프로토타입



Array 생성자는 다양한 메소드를 갖는다.  
prototype도 그중 하나

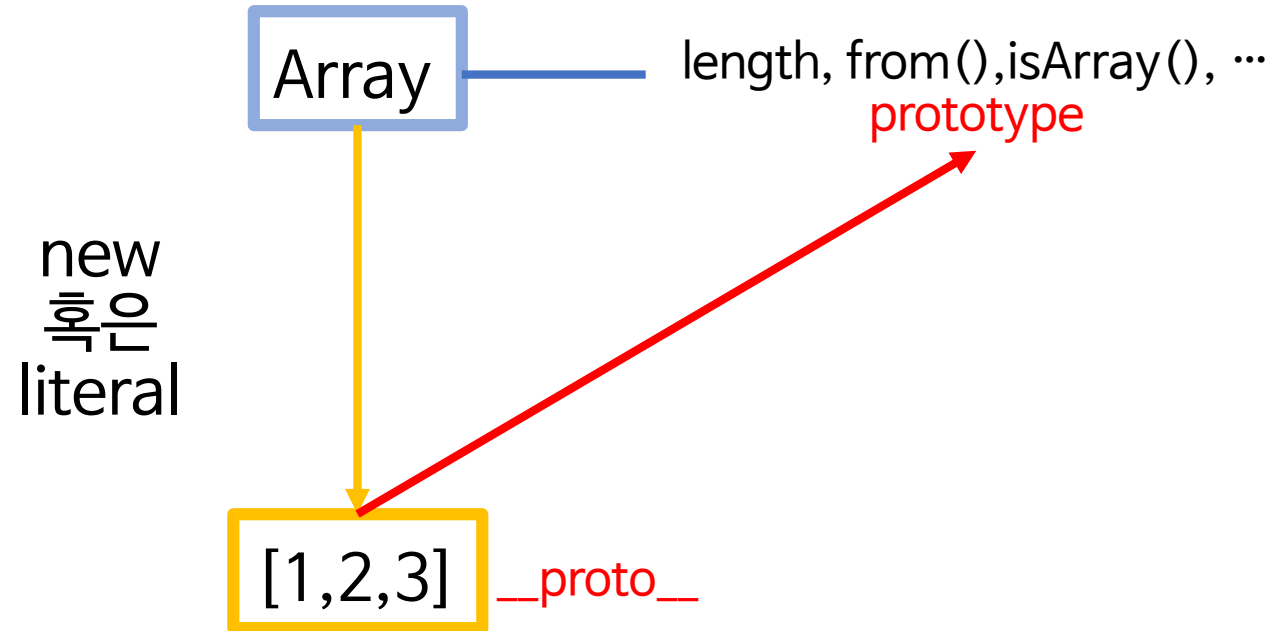


# 1. 프로토타입



Array 생성자의 prototype이  
배열 인스턴스의 \_\_proto\_\_를 가리키게 된다.

# 1. 프로토타입



인스턴스에서 \_\_proto\_\_는 생략이 가능하다.  
인스턴스는 자신의 메소드처럼 prototype을 사용할 수 있다.

# 1. 프로토타입

```
> console.dir(Array)
```

```
▼ f Array() ⓘ
```

```
  arguments: (...)
```

```
  caller: (...)
```

```
  ▶ from: f from()
```

```
  ▶ isArray: f isArray()
```

```
  length: 1
```

```
  name: "Array"
```

```
  ▶ of: f of()
```

```
  ▶ prototype: [constructor: f, concat: f, copyWithin: f, fill: f, find: f, ...]
```

```
    Symbol(Symbol.species): (...)
```

```
  ▶ get Symbol(Symbol.species): f [Symbol.species]()
```

```
  ▶ __proto__: f ()
```

```
  ▶ [[Scopes]]: Scopes[0]
```

## 1. 프로토타입

```
> console.dir([1,2,3]);  
▼ Array(3) ⓘ  
  0: 1  
  1: 2  
  2: 3  
  length: 3  
  ► __proto__: Array(0)
```

# 1. 프로토타입

```
> console.dir(Array)
▼ f Array()
  arguments: (...)
  caller: (...)
  ▶ from: f from()
  ▶ isArray: f isArray()
  length: 1
  name: "Array"
  ▶ of: f of()
  ▼ prototype: Array(0)
    ▶ concat: f concat()
    ▶ constructor: f Array()
    ▶ copyWithin: f copyWithin()
    ▶ entries: f entries()
    ▶ every: f every()
    ▶ fill: f fill()
    ▶ filter: f filter()
    ▶ find: f find()
    ▶ findIndex: f findIndex()
    ▶ flat: f flat()
    ▶ flatMap: f flatMap()
    ▶ forEach: f forEach()
    ▶ includes: f includes()
    ▶ indexOf: f indexOf()
    ▶ join: f join()
    ▶ keys: f keys()
    ▶ lastIndexOf: f lastIndexOf()
    length: 0
    ▶ map: f map()
    ▶ pop: f pop()
    ▶ push: f push()
    ▶ reduce: f reduce()
```

```
> console.dir([1,2,3]);
▼ Array(3)
  0: 1
  1: 2
  2: 3
  length: 3
  ▼ proto : Array(0)
    ▶ concat: f concat()
    ▶ constructor: f Array()
    ▶ copyWithin: f copyWithin()
    ▶ entries: f entries()
    ▶ every: f every()
    ▶ fill: f fill()
    ▶ filter: f filter()
    ▶ find: f find()
    ▶ findIndex: f findIndex()
    ▶ flat: f flat()
    ▶ flatMap: f flatMap()
    ▶ forEach: f forEach()
    ▶ includes: f includes()
    ▶ indexOf: f indexOf()
    ▶ join: f join()
    ▶ keys: f keys()
    ▶ lastIndexOf: f lastIndexOf()
    length: 0
    ▶ map: f map()
    ▶ pop: f pop()
    ▶ push: f push()
```

## 1. 프로토타입

`[constructor].prototype`

`[instance].__proto__`

`[instance]`

`Object.getPrototypeOf([instance])`

4가지 방식은 모두 같은 prototype을 가리킨다

# 1. 프로토타입

```
function User(id, name) {  
  this.id = id;  
  this.name = name;  
}  
  
const user = new User(0, "jinyoung");  
  
const user1 = new user.__proto__.constructor(1, "j");  
  
const user2 = new user.constructor(2, "y");  
  
const userProtoType = Object.getPrototypeOf(user);  
  
const user3 = new userProtoType.constructor("3", "o");
```

user, user1, user2, user3  
모두 User의 인스턴스이다.

## 2. 프로토타입 체이닝

Static method, property :  
소속여부의 확인, 소속부여 등 공동체적인 작업

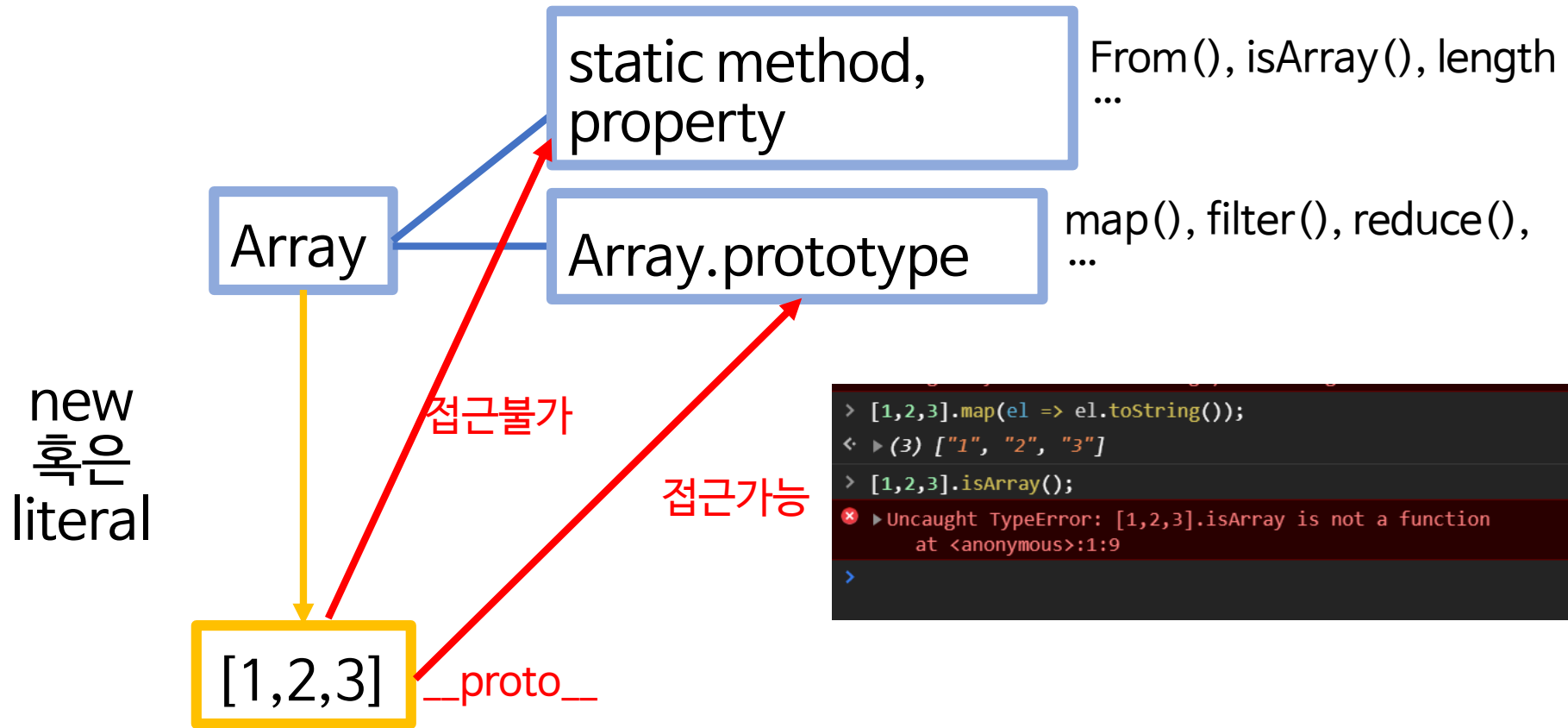
```
> console.dir(Array)
▼ f Array()
  arguments: (...)
  caller: (...)
  ▶ from: f from()
  ▶ isArray: f isArray()
  length: 1
  name: "Array"
  ▶ of: f of()
  ▼ prototype: Array(0)
    ▶ concat: f concat()
    ▶ constructor: f Array()
    ▶ copyWithin: f copyWithin()
    ▶ entries: f entries()
    ▶ every: f every()
    ▶ fill: f fill()
    ▶ filter: f filter()
    ▶ find: f find()
    ▶ findIndex: f findIndex()
    ▶ flat: f flat()
    ▶ flatMap: f flatMap()
    ▶ forEach: f forEach()
    ▶ includes: f includes()
    ▶ indexOf: f indexOf()
    ▶ join: f join()
    ▶ keys: f keys()
    ▶ lastIndexOf: f lastIndexOf()
    length: 0
    ▶ map: f map()
    ▶ pop: f pop()
    ▶ push: f push()
    ▶ reduce: f reduce()
```

```
> console.dir([1,2,3]);
▼ Array(3)
  0: 1
  1: 2
  2: 3
  length: 3
  ▼ proto : Array(0)
    ▶ concat: f concat()
    ▶ constructor: f Array()
    ▶ copyWithin: f copyWithin()
    ▶ entries: f entries()
    ▶ every: f every()
    ▶ fill: f fill()
    ▶ filter: f filter()
    ▶ find: f find()
    ▶ findIndex: f findIndex()
    ▶ flat: f flat()
    ▶ flatMap: f flatMap()
    ▶ forEach: f forEach()
    ▶ includes: f includes()
    ▶ indexOf: f indexOf()
    ▶ join: f join()
    ▶ keys: f keys()
    ▶ lastIndexOf: f lastIndexOf()
    length: 0
    ▶ map: f map()
    ▶ pop: f pop()
    ▶ push: f push()
```

(prototype) method : 메소드



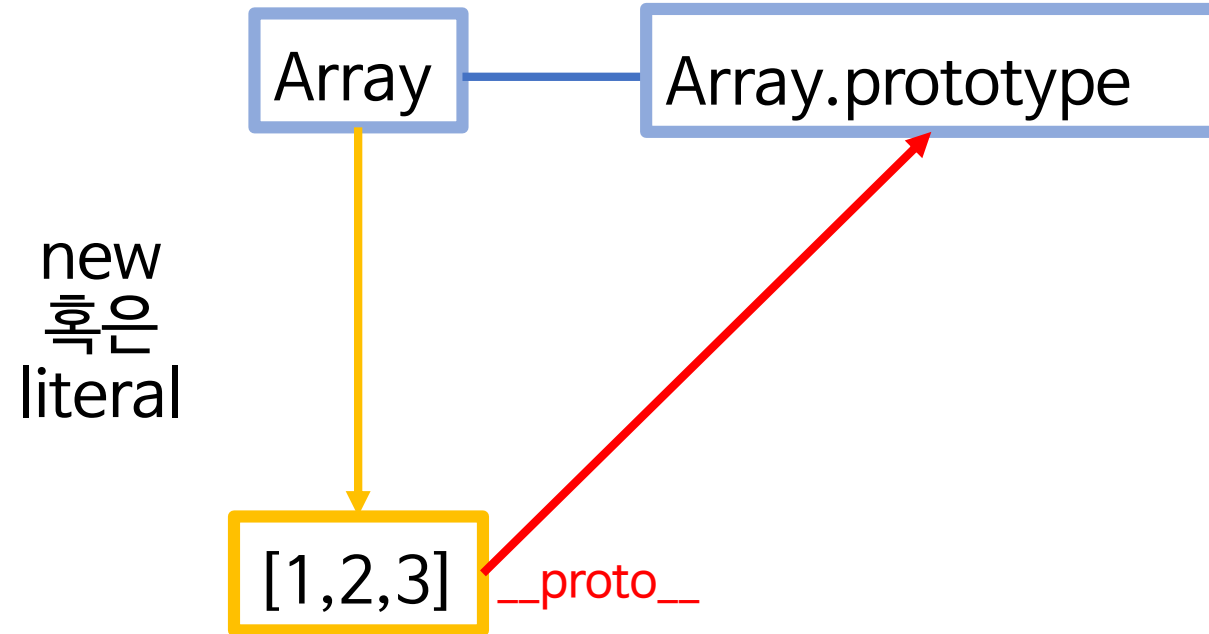
## 2. 프로토타입 체이닝



인스턴스는 클래스의 static method, property에 접근할 수 없다.

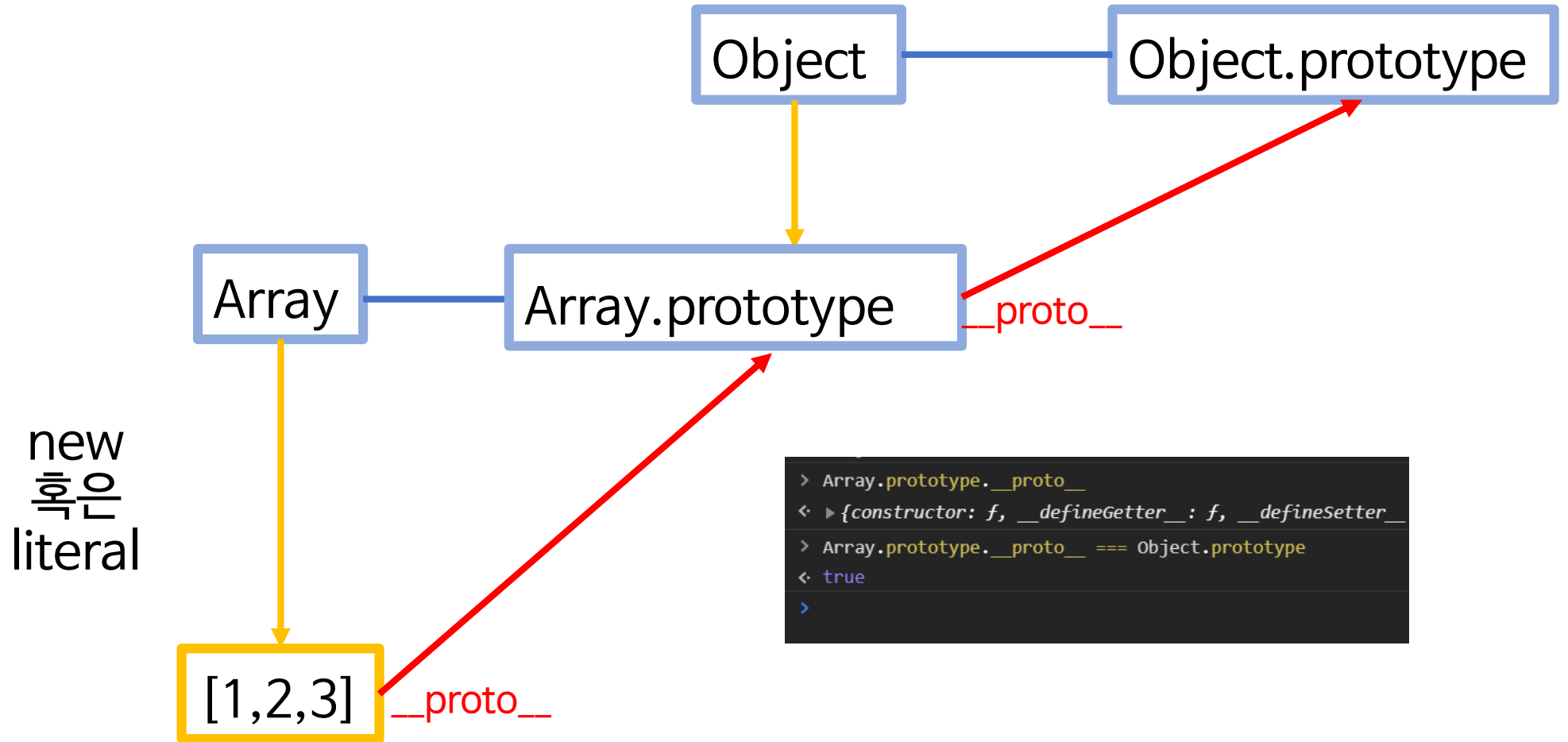
## 2. 프로토타입 체이닝

```
> Array.prototype  
< ▶ [constructor: f, concat: f, copyWithin: f, fill: f, find: f, ...]  
  
> typeof(Array.prototype)  
< "object"
```



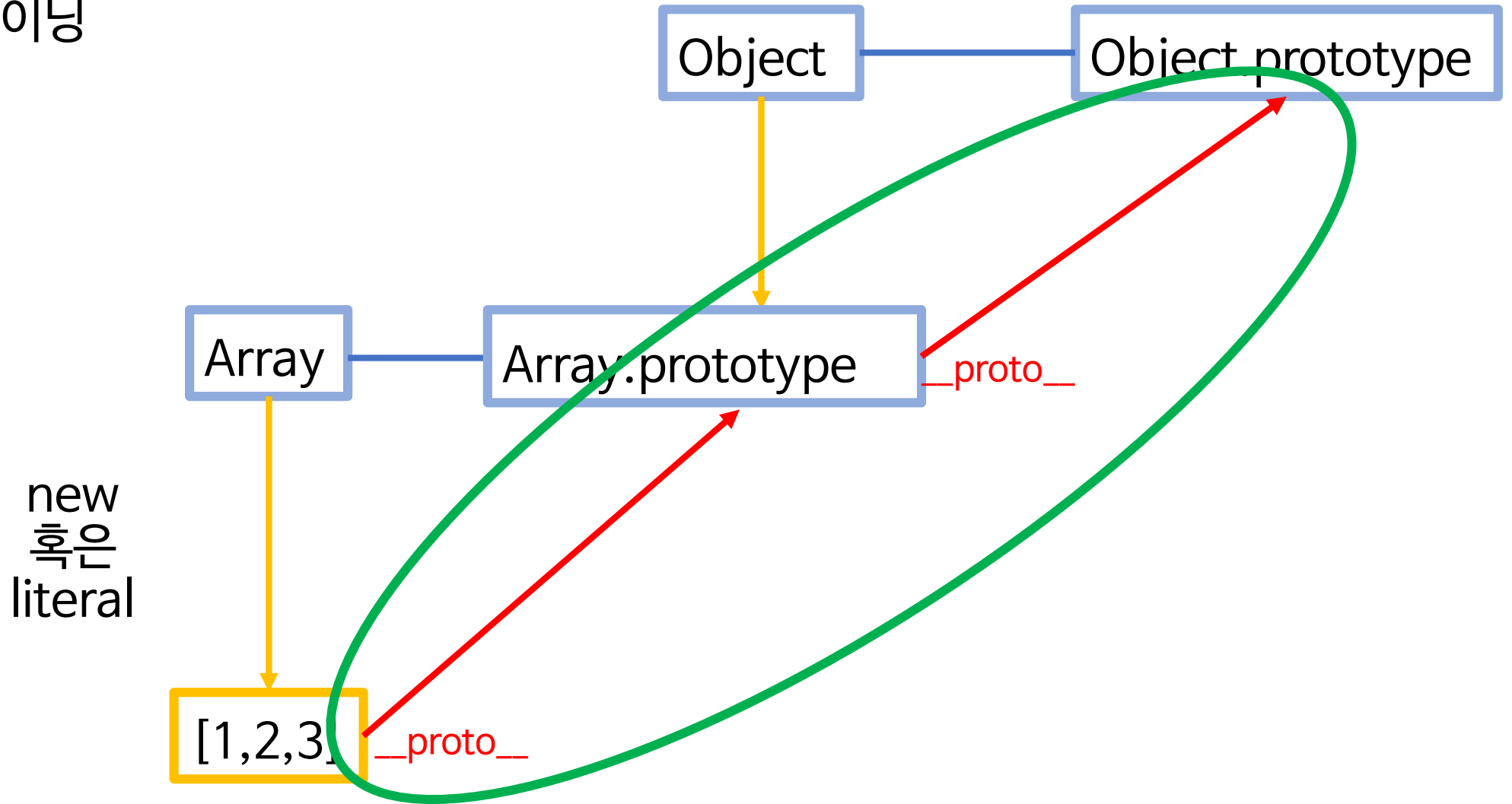
Prototype 또한 객체이다.

## 2. 프로토타입 체이닝



Prototype 또한 객체이다. == Object 생성자로 생성된 인스턴스이다.

## 2. 프로토타입 체이닝



`__proto__`를 통해 객체가 연결된 관계를  
프로토타입 체이닝이라고 한다

## 2. 프로토타입 체이닝

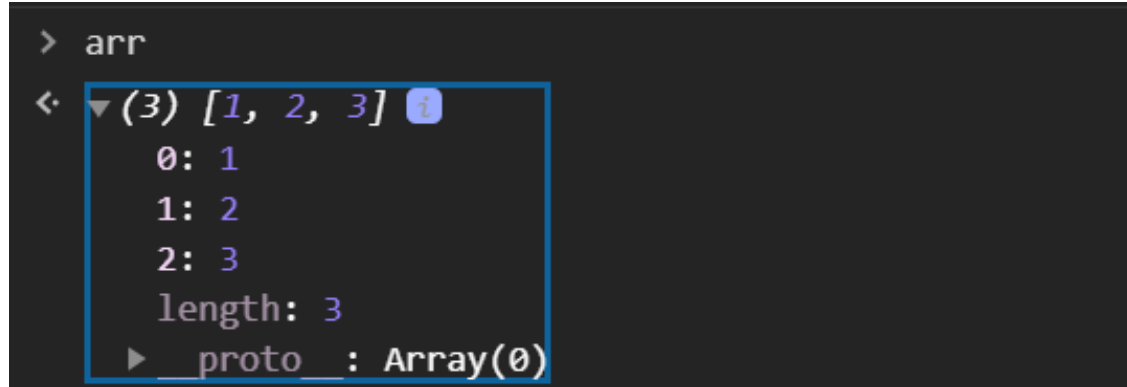
Javascript에서 프로토타입을 사용하는 방법

```
> const arr = [1,2,3];  
↵ undefined  
  
> arr.toString()  
↵ "1,2,3"
```

toString()을 사용하고 싶다!

## 2. 프로토타입 체이닝


Javascript에서 프로토타입을 사용하는 방법



1. 객체 자체 메소드를 탐색한다  
=> 없음

## 2. 프로토타입 체이닝

### Javascript에서 프로토타입을 사용하는 방법

```
> arr.__proto__  
◀ [constructor: f, concat: f, copyWithin: f, fill: f, find: f, ...]   
  ▶ concat: f concat()  
  ▶ constructor: f Array()  
  ▶ copyWithin: f copyWithin()  
  ▶ entries: f entries()  
  ▶ every: f every()  
  ▶ fill: f fill()  
  ▶ filter: f filter()  
  ▶ find: f find()  
  ▶ findIndex: f findIndex()  
  ▶ flat: f flat()  
  ▶ flatMap: f flatMap()  
  ▶ forEach: f forEach()  
  ▶ includes: f includes()  
  ▶ indexOf: f indexOf()  
  ▶ join: f join()  
  ▶ keys: f keys()
```

2. arr.\_\_proto\_\_를 탐색한다  
=> 없음

## 2. 프로토타입 체이닝

### Javascript에서 프로토타입을 사용하는 방법

```
> arr.__proto__.__proto__  
◀ {constructor: f, __defineGetter__: f, __defineSetter__: f, hasOwnProperty: f, __lookupGetter__: f, ...}  
  ▶ constructor: f Object()  
  ▶ hasOwnProperty: f hasOwnProperty()  
  ▶ isPrototypeOf: f isPrototypeOf()  
  ▶ propertyIsEnumerable: f propertyIsEnumerable()  
  ▶ toLocaleString: f toLocaleString()  
  ▶ toString: f toString()  
  ▶ valueOf: f valueOf()  
  ▶ __defineGetter__: f __defineGetter__()  
  ▶ __defineSetter__: f __defineSetter__()  
  ▶ __lookupGetter__: f __lookupGetter__()  
  ▶ __lookupSetter__: f __lookupSetter__()  
  ▶ get __proto__: f __proto__()  
  ▶ set __proto__: f __proto__()
```

3. arr.\_\_proto\_\_.\_\_proto\_\_를 탐색한다  
=> toString() 발견!



## 2. 프로토타입 체이닝

Javascript에서 프로토타입을 사용하는 방법

```
> arr.__proto__.__proto__.__proto__  
◀ null
```

메소드를 찾을때까지 **\_\_proto\_\_**를 재귀적으로 탐색하고,  
null을 만나면 종료한다.

## 2. 프로토타입 체이닝

```
> const number = 1;
< undefined

> number.toString();
< "1"

> number.__proto__.__proto__.__proto__
< null

> '123'.toString();
< "123"

> '123'.__proto__.__proto__.__proto__
< null
```

Javascript의 모든 데이터 타입은 프로토타입 체이닝 구조를 띄고 있다.

## 2. 프로토타입 체이닝

ES6 클래스 는 프로토타입 패턴을 기반으로 한 문법.

프로토타입 체이닝을 이용하여  
클래스의 상속(extends, super)을 구현할 수 있다.

참고자료

인프런 강의 : Javascript 핵심 개념 알아보기 - JS FLOW(정재남)  
(<https://www.inflearn.com/course/%ED%95%B5%EC%8B%AC%EA%B0%9C%EB%85%90-javascript-flow/dashboard>)

Mozilla 공식 사이트

[https://developer.mozilla.org/ko/docs/Web/JavaScript/Guide/Inheritance\\_and\\_the\\_prototype\\_chain](https://developer.mozilla.org/ko/docs/Web/JavaScript/Guide/Inheritance_and_the_prototype_chain)