

Prototype Chaining

목차

1. 프로토타입
2. 프로토타입 체이닝
3. 클래스 상속

1. 프로토타입

자바스크립트는 프로토타입 기반 언어이다

자바스크립트의 객체는 프로토타입을 갖는다

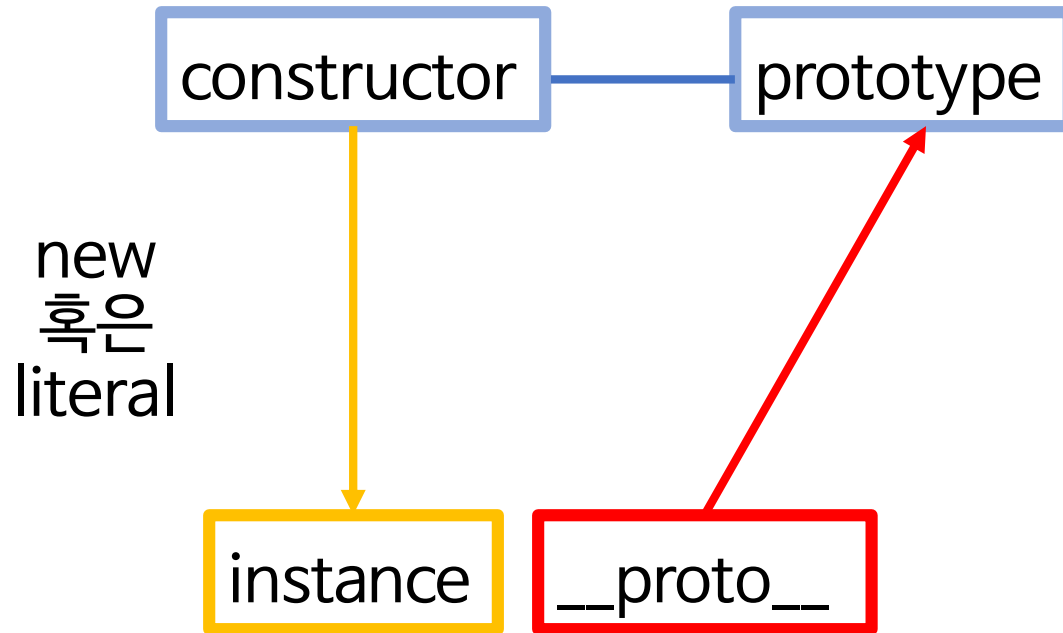
prototype, constructor, __proto__

1. 프로토타입



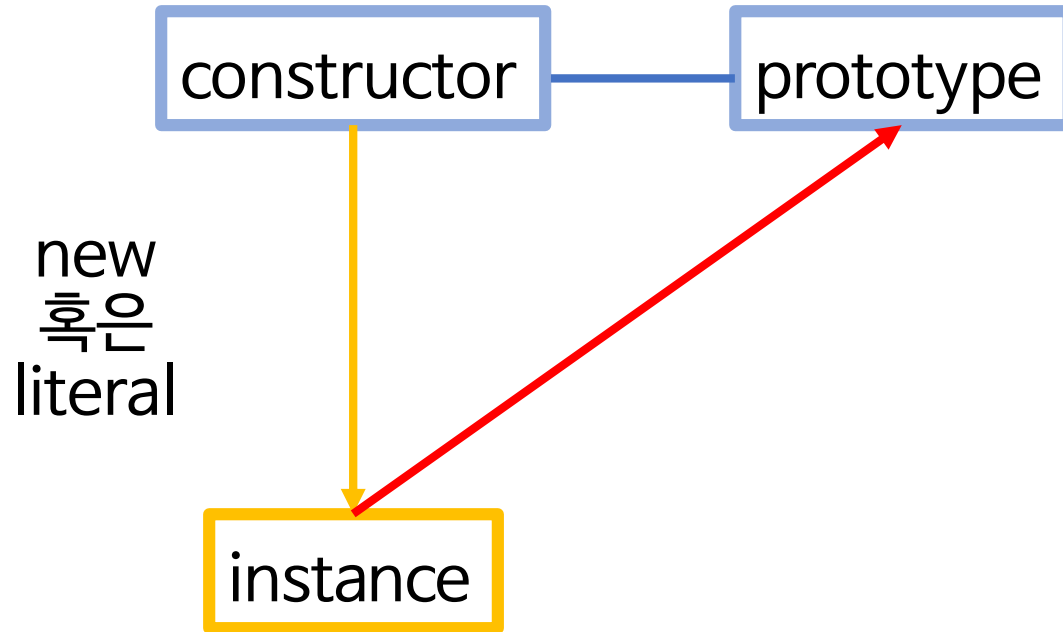
constructor라는 생성자 함수는 prototype을 갖는다

1. 프로토타입



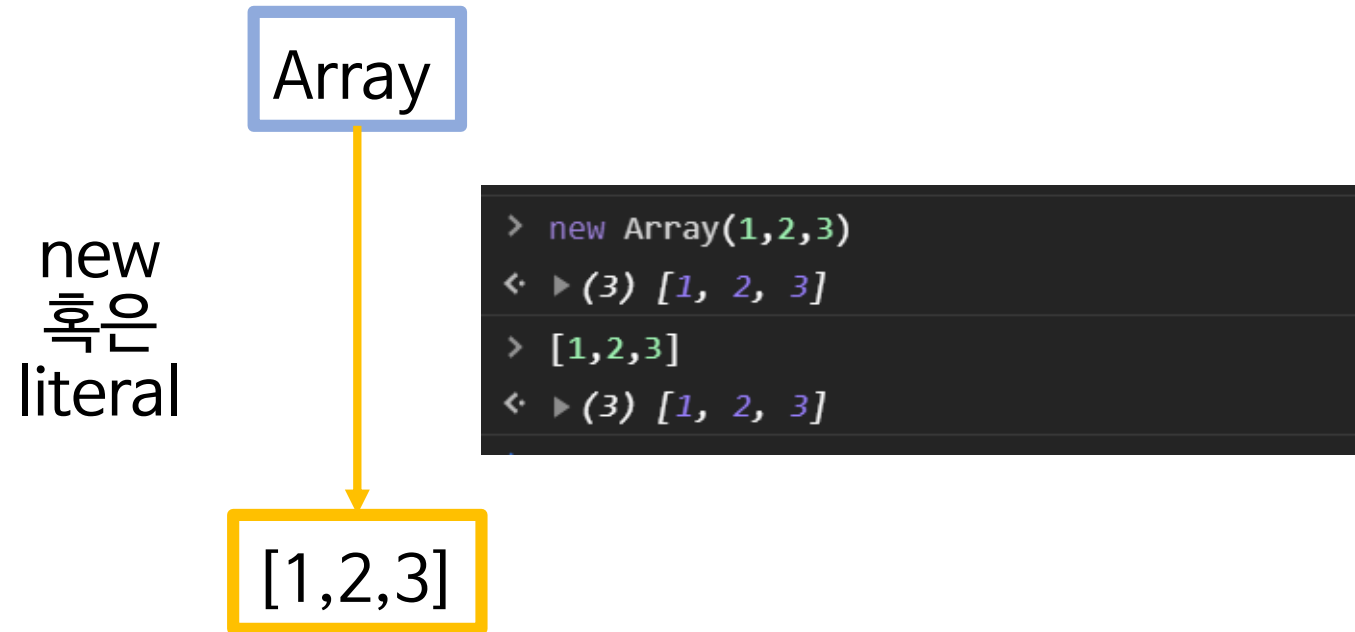
New 혹은 리터럴 ({}, [])을 통해 인스턴스를 생성하면
인스턴스의 `__proto__`는 `prototype`을 가리키게 된다.
즉 생성자의 `prototype`과 인스턴스의 `__proto__`는 같다.

1. 프로토타입



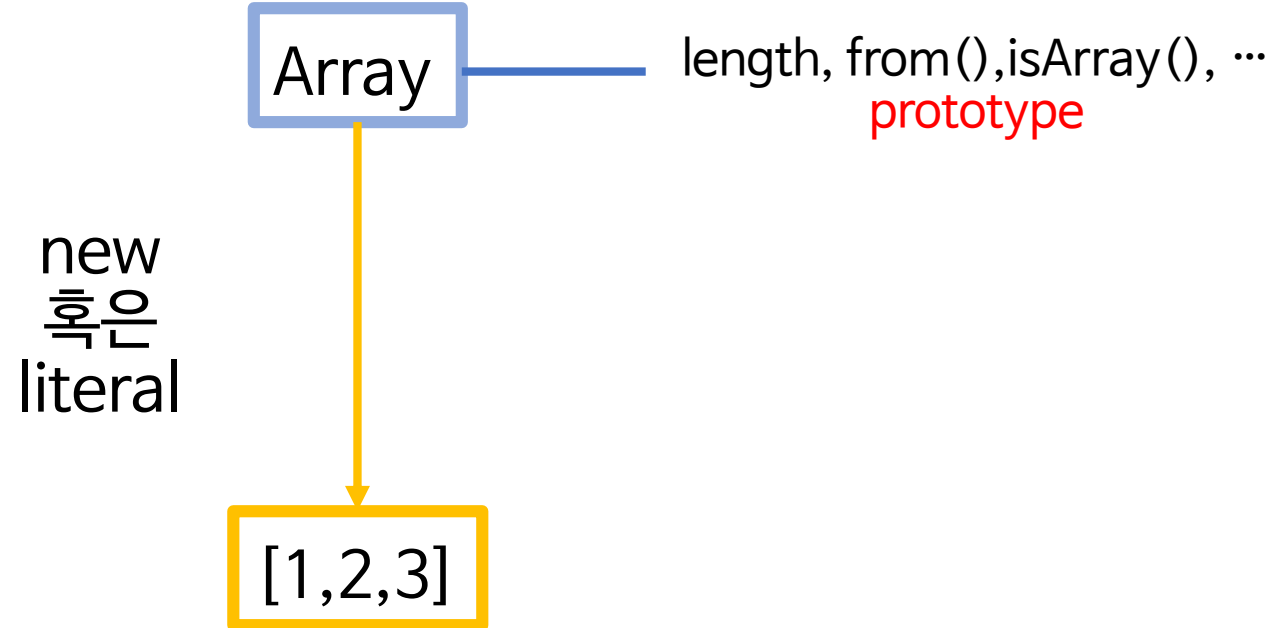
Instance에서 prototype을 쓸 때, `__proto__`는 생략가능하다.
(실제로는 아니지만 그런것처럼 작동한다)

1. 프로토타입



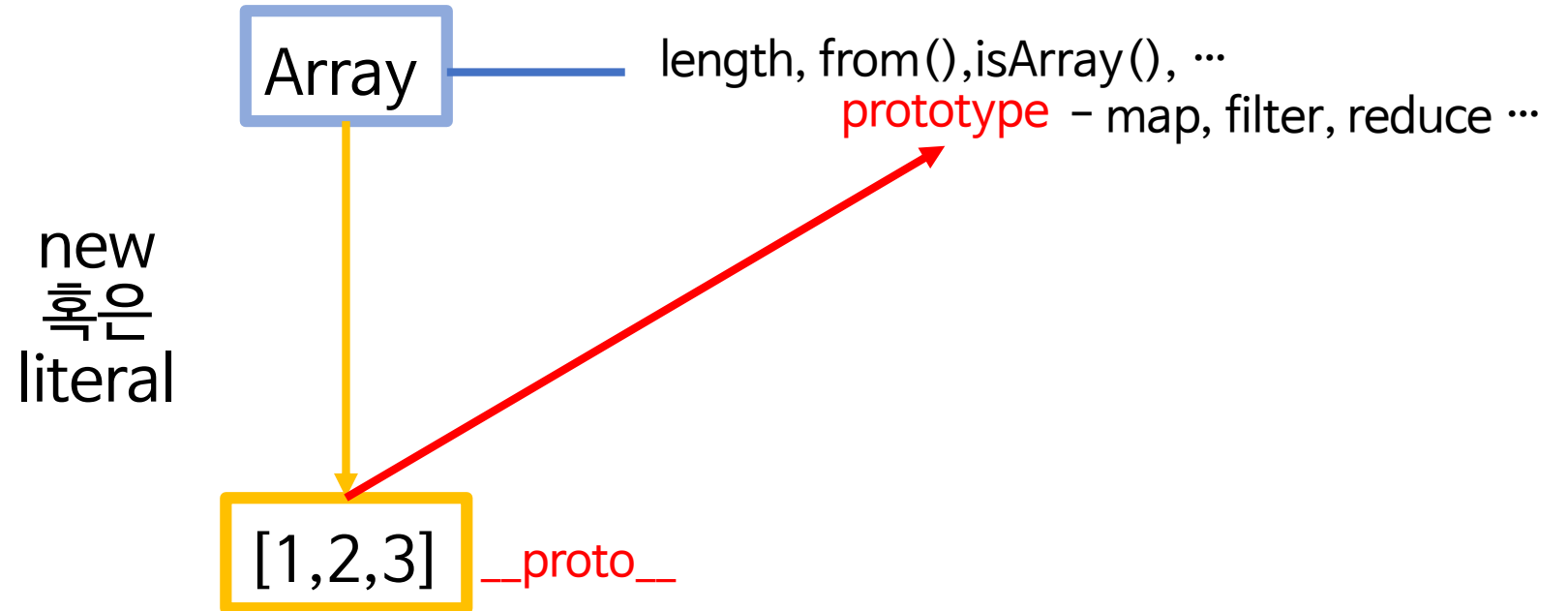
Array생성자를 통해 배열을 생성한다.

1. 프로토타입



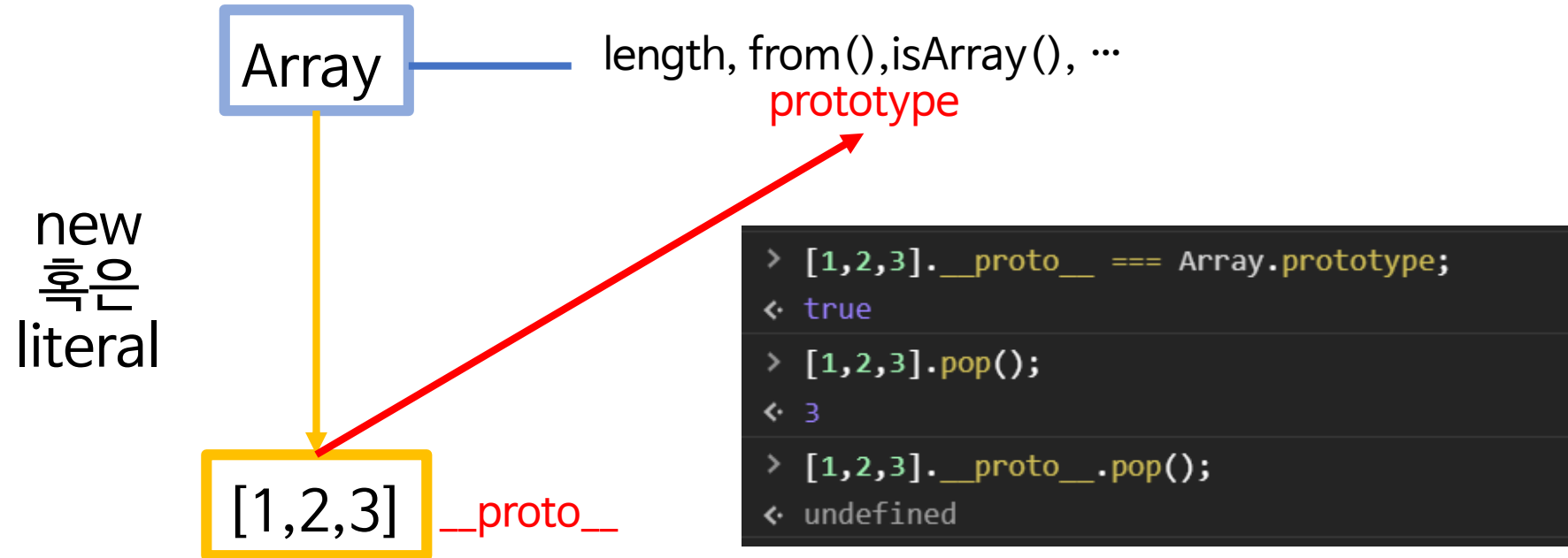
Array 생성자는 다양한 메소드를 갖는다.
prototype도 그중 하나

1. 프로토타입



배열 인스턴스의 `__proto__`가
Array 생성자의 prototype을 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

`[constructor].prototype.constructor === constructor`

```
> Array.prototype.constructor === Array  
◀ true
```

prototype은 생성자함수를 내장하고 있다.

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

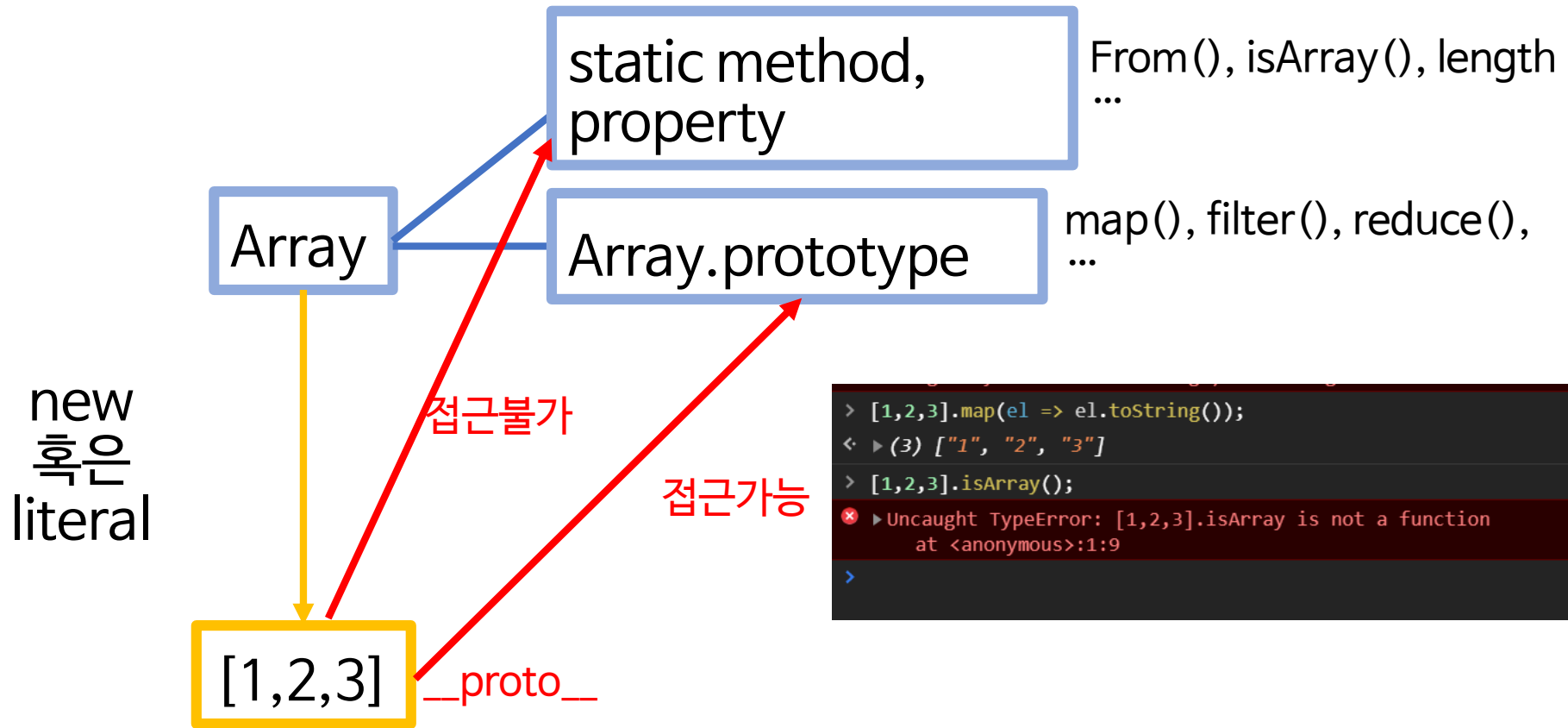
Static method, property :
소속여부의 확인, 소속부여 등 공동체적인 작업

```
> 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

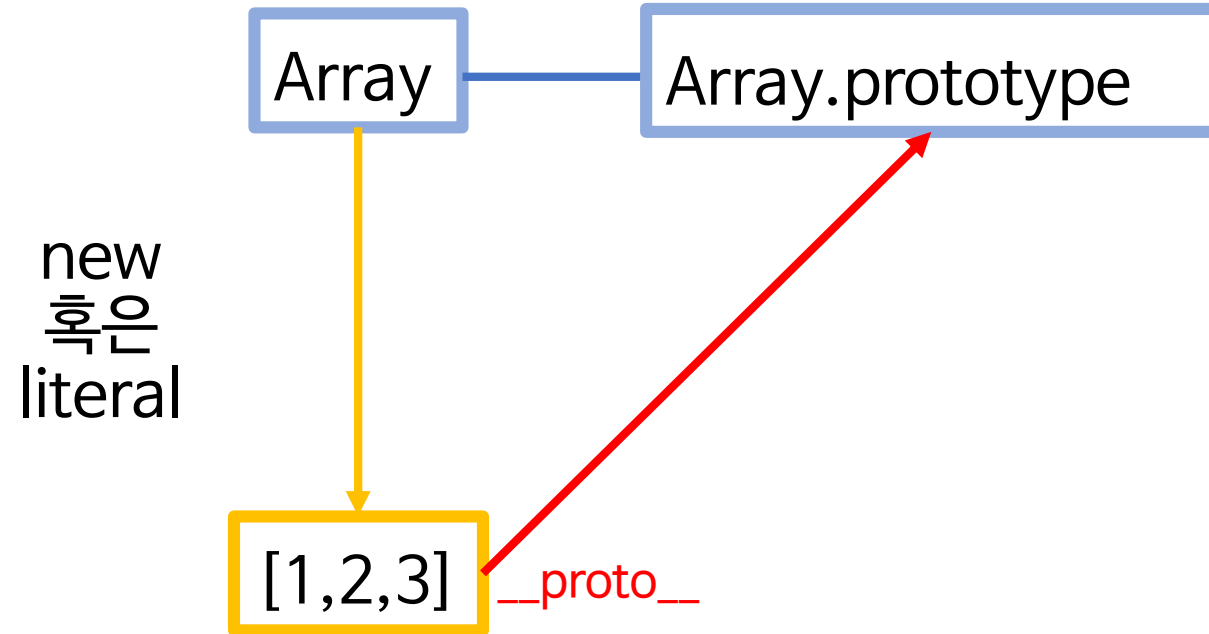
2. 프로토타입 체이닝



인스턴스는 클래스의 static method, property에 접근할 수 없다.

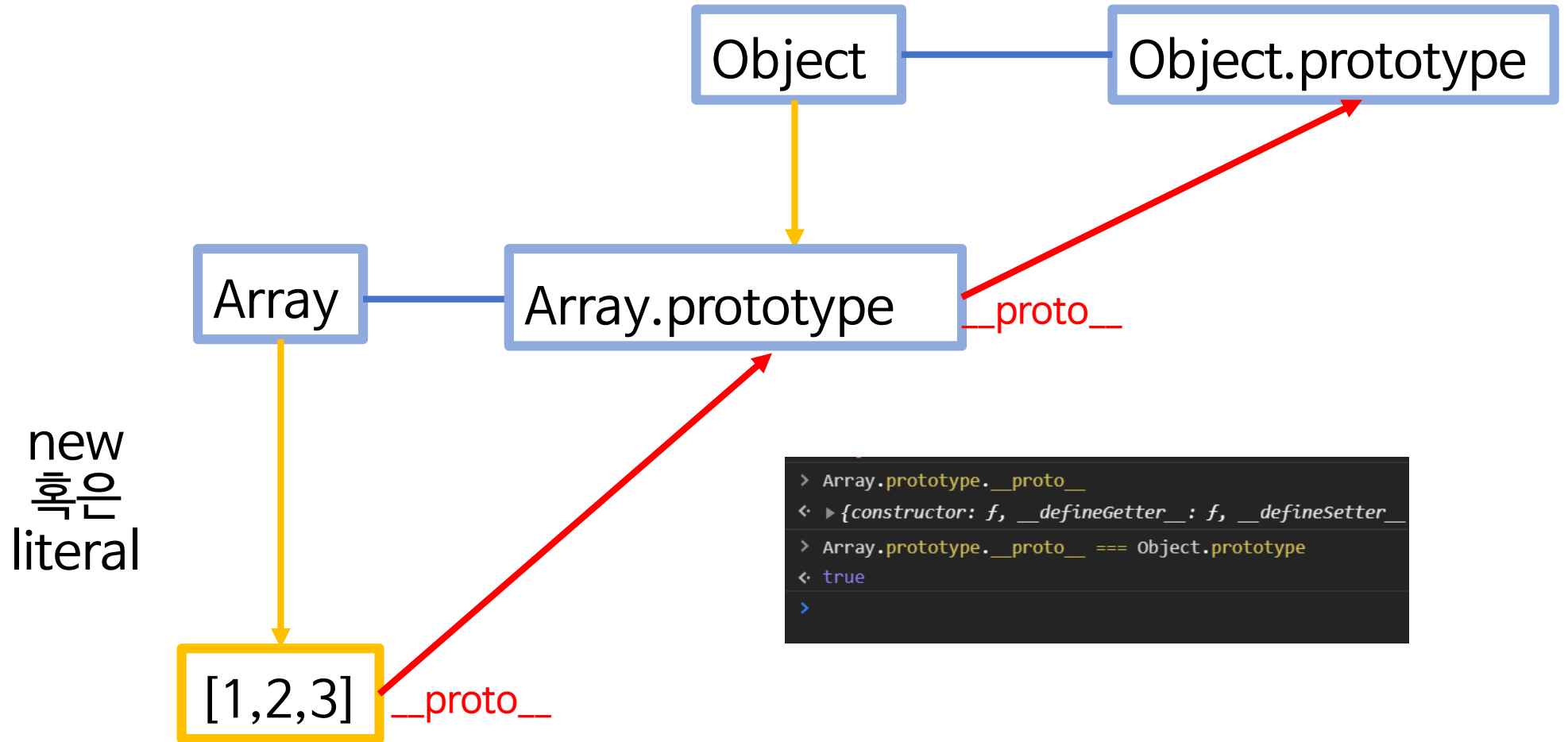
2. 프로토타입 체이닝

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



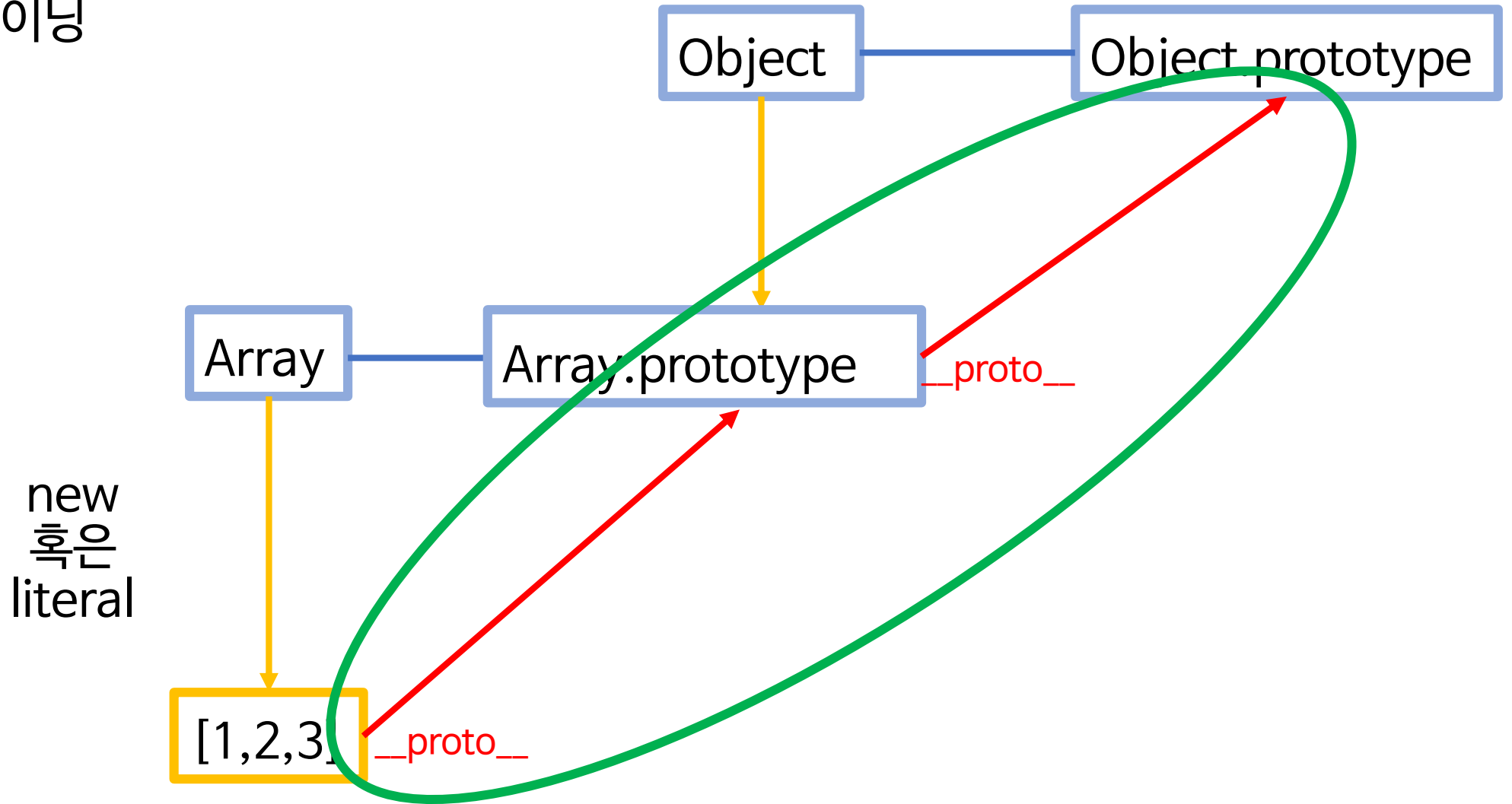
Prototype 또한 객체이다.

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

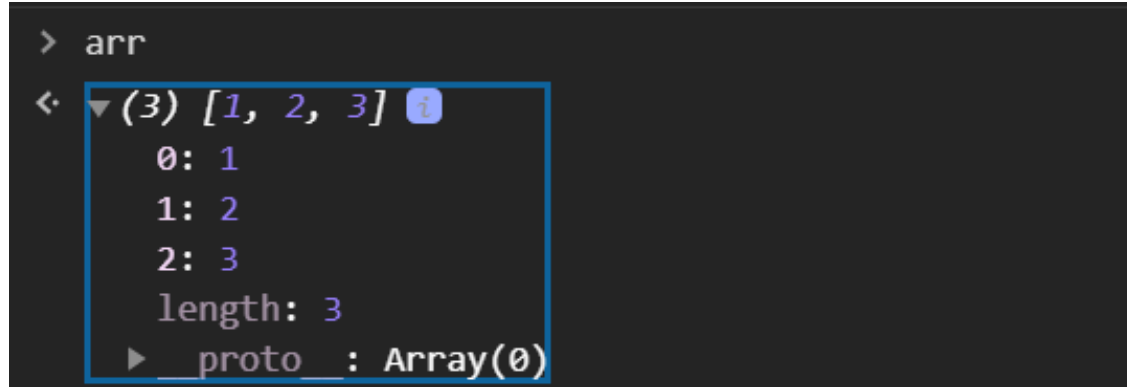
Javascript에서 프로토타입을 사용하는 방법

```
> 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.

3. 클래스 상속

ES6 클래스 는 프로토타입 패턴을 기반으로 한 문법. (syntactic sugar)

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

3. 클래스 상속

```
function Person(name, age) {  
  this.name = name;  
  this.age = age;  
}  
  
Person.prototype.setAge = function (age) {  
  this.age = age;  
};  
  
Person.prototype.getAge = function () {  
  return this.age;  
};
```

Prototype

```
class Person {  
  constructor(name, age) {  
    this.name = name;  
    this.age = age;  
  }  
  
  setAge(age) {  
    this.age = age;  
  }  
  
  getAge() {  
    return this.age;  
  }  
}
```

Class

3. 클래스 상속

```
> console.dir(Person);
```

```
▼ f Person(name, age) ⓘ  
  arguments: null  
  caller: null  
  length: 2  
  name: "Person"  
  ▼ prototype:  
    ▶ getAge: f ()  
    ▶ setAge: f (age)  
    ▶ constructor: f Person(name, age)  
    ▶ __proto__: Object  
  ▶ __proto__: f ()  
    [[FunctionLocation]]: VM175:1  
  ▶ [[Scopes]]: Scopes[1]
```

Prototype

```
> console.dir(PersonClass);
```

```
▼ class PersonClass ⓘ  
  arguments: (...)  
  caller: (...)  
  length: 2  
  name: "PersonClass"  
  ▼ prototype:  
    ▶ constructor: class PersonClass  
    ▶ getAge: f getAge()  
    ▶ setAge: f setAge(age)  
    ▶ __proto__: Object  
  ▶ __proto__: f ()  
    [[FunctionLocation]]: VM384:2  
  ▶ [[Scopes]]: Scopes[2]
```

Class

3. 클래스 상속

클래스 상속을 프로토타입으로 구현해보자

```
> class Person {  
  constructor(name, age) {  
    this.name = name;  
    this.age = age;  
  }  
  
  setAge(age) {  
    this.age = age;  
  }  
  
  getAge() {  
    return this.age;  
  }  
}
```

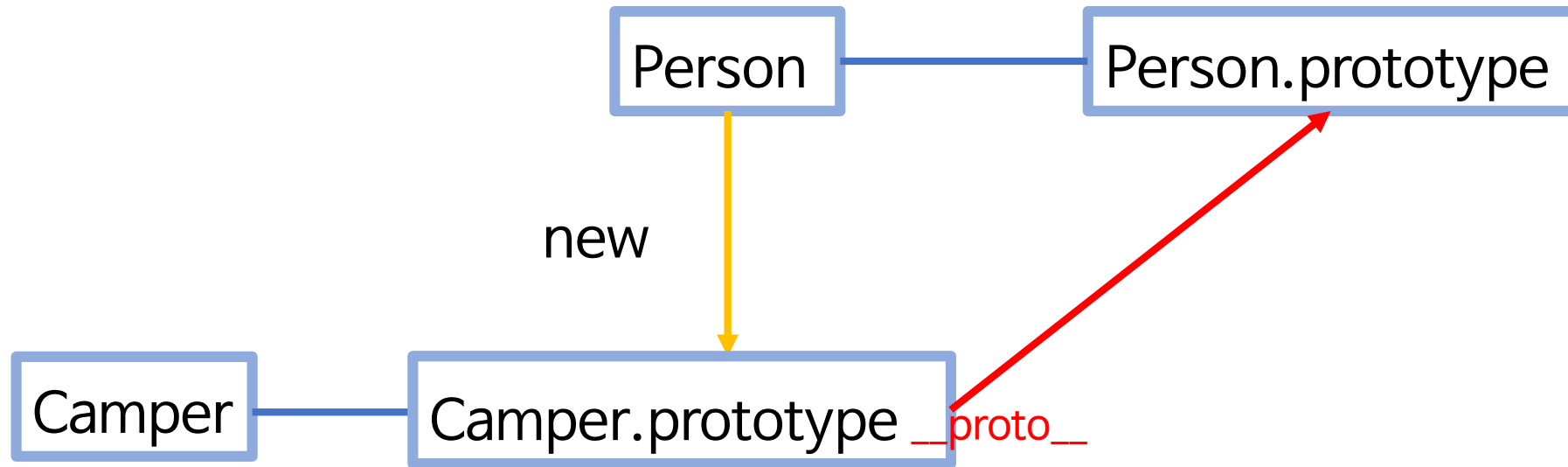
```
class Camper extends Person {  
  constructor(name, age, id) {  
    super(name, age);  
    this.id = id;  
  }  
  
  setId(id) {  
    this.id = id;  
  }  
  
  getId() {  
    return this.id;  
  }  
}
```

3. 클래스 상속

```
> console.dir(Camper);  
▼ class Camper ⓘ  
  arguments: (...)  
  caller: (...)  
  length: 3  
  name: "Camper"  
  ▼ prototype: Person  
    ▶ constructor: class Camper  
    ▶ getId: f getId()  
    ▶ setId: f setId(id)  
    ▶ __proto__: Object  
    ▶ proto : class Person  
      [[FunctionLocation]]: VM872:17  
    ▶ [[Scopes]]: Scopes[2]
```

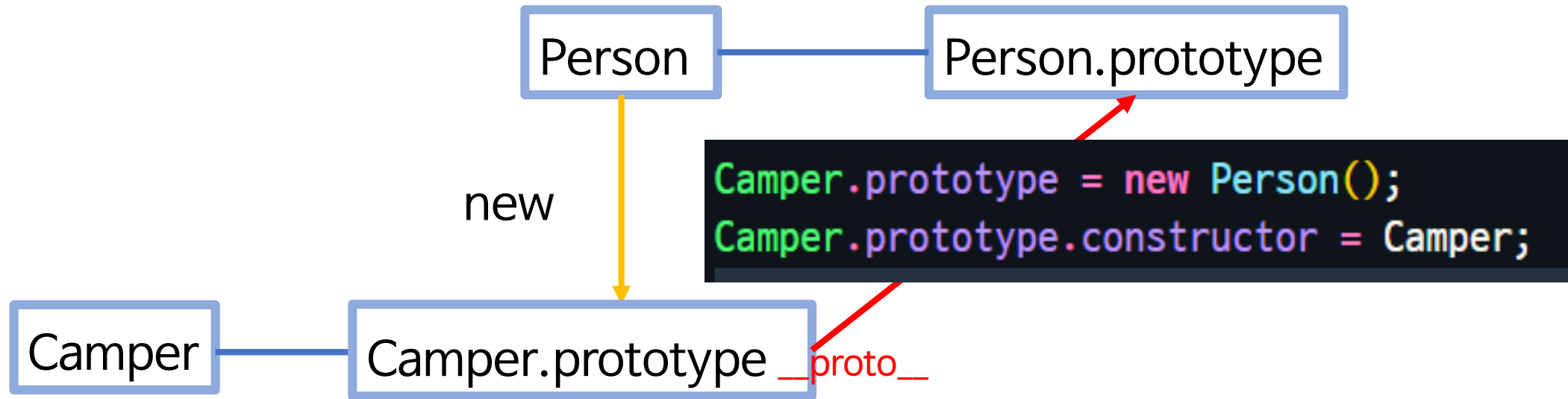
상속받은 클래스는 부모를 prototype으로 갖는다

3. 클래스 상속



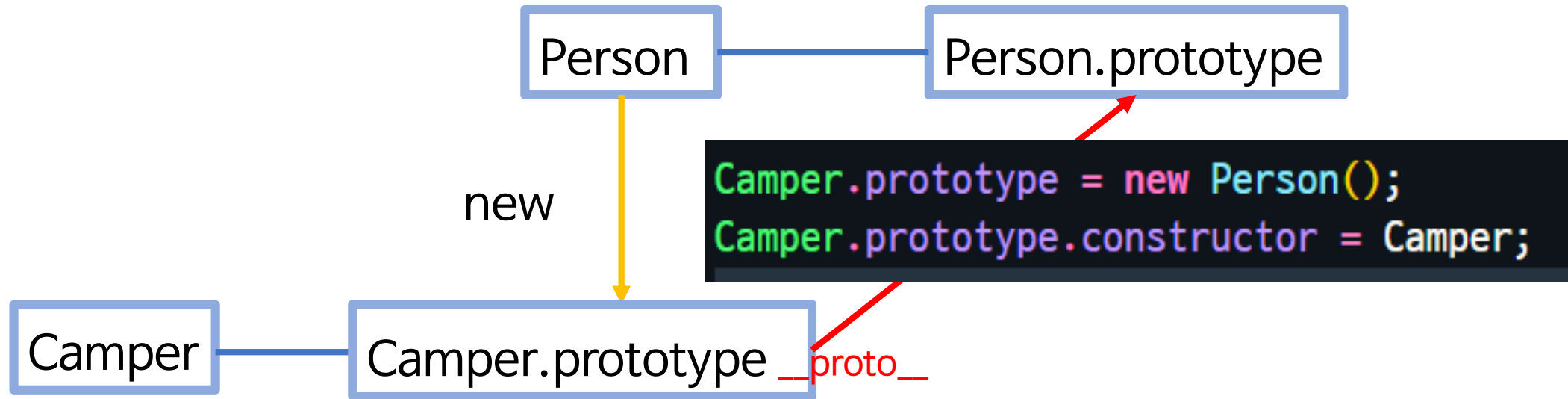
상속받은 클래스는 부모를 prototype으로 갖는다

3. 클래스 상속



상속받은 클래스는 부모를 prototype으로 갖는다

3. 클래스 상속



상속받은 클래스는 부모를 prototype으로 갖는다

3. 클래스 상속

```
> function Person(name, age) {  
  this.name = name;  
  this.age = age;  
}  
  
Person.prototype.setAge = function (age) {  
  this.age = age;  
};  
  
Person.prototype.getAge = function () {  
  return this.age;  
};  
  
function Camper(id) {  
  this.id = id;  
}  
  
Camper.prototype.setId = function(id){  
  this.id = id;  
}  
  
Camper.prototype.getId = function(){  
  return this.id;  
}
```

```
> console.dir(Camper);  
  
▼ f Camper(id) ⓘ  
  arguments: null  
  caller: null  
  length: 1  
  name: "Camper"  
  ▶ prototype: {setId: f, getId: f, constructor: f}  
  ▶ __proto__: f ()  
    [[FunctionLocation]]: VM2223:14  
  ▶ [[Scopes]]: Scopes[1]
```

3. 클래스 상속

```
> Camper.prototype = new Person();  
< ▶ Person {name: undefined, age: undefined}  
  
> console.dir(Camper);  
  
▼ f Camper(id) ⓘ  
  arguments: null  
  caller: null  
  length: 1  
  name: "Camper"  
  ▶ prototype: Person {name: undefined, age: undefined}  
  ▶ __proto__: f ()  
    [[FunctionLocation]]: VM2223:14  
  ▶ [[Scopes]]: Scopes[1]
```

```
> Camper.prototype.constructor = Camper;  
< f Camper(id) {  
  this.id = id;  
}  
  
> console.dir(Camper);  
  
▼ f Camper(id) ⓘ  
  arguments: null  
  caller: null  
  length: 1  
  name: "Camper"  
  ▼ prototype: Person  
    age: undefined  
    ▶ constructor: f Camper(id)  
      name: undefined  
    ▶ __proto__: Object  
  ▶ __proto__: f ()  
    [[FunctionLocation]]: VM2223:14  
  ▶ [[Scopes]]: Scopes[1]
```

3. 클래스 상속

```
> console.dir(Camper);
```

▼ class Camper ⓘ

arguments: (...)

caller: (...)

length: 3

name: "Camper"

▼ prototype: Person

▶ constructor: class Camper

▶ getId: f getId()

▶ setId: f setId(id)

▶ __proto__: Object

▶ proto : class Person

[[FunctionLocation]]: VM872:17

▶ [[Scopes]]: Scopes[2]

```
> Camper.prototype.constructor = Camper;
```

```
< f Camper(id) {  
  this.id = id;  
}
```

```
> console.dir(Camper);
```

▼ f Camper(id) ⓘ

arguments: null

caller: null

length: 1

name: "Camper"

▼ prototype: Person

age: undefined

▶ constructor: f Camper(id)

name: undefined

▶ __proto__: Object

▶ __proto__: f ()

[[FunctionLocation]]: VM2223:14

▶ [[Scopes]]: Scopes[1]

3. 클래스 상속

감사합니다!

참고자료

인프런

Javascript 핵심 개념 알아보기 - JS FLOW (정재남)

Mozilla - 상속과 프로토타입

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