

jsFlow 2017. 9. 9. gomugom

---

# JAVASCRIPT 흐름 파악하기

게임회사 웹서비스 개발

4년차 front-end developer

정재남

jquery / javascript 강의형 스터디 - 5회

ReactJS 강의형 스터디 - 7회

기타 다양한 스터디 지속중

시간 확신  
이해 활용

# CONTENTS

1. 데이터 탑입 30분

break 10분

2. Function 50분

break 10분

3. this 20분

4. closure 40분

break 10분

5. prototype 35분

6. class 25분

jsFlow 2017. 9. 9. gomugom

---

# 1. DATA TYPES

## Primitive Type

값을 그대로 할당

Number  
String  
Boolean  
null  
undefined

## Reference Type — Object

값이 저장된 주소값을 할당 (참조)

Array  
Function  
RegExp

```
var a;
```

변수명	a	3		
주소	@313	2		

주소	...	313	314	315	316	317	...
데이터		-	1				

```
var a;  
a = 10;
```

변수명	a	1		
주소	@313	2		

주소	...	313	314	315	316	317	...
데이터		10	3				

```
var a;           var b = 'abc';  
a = 10;
```

변수명	a	b	3		
주소	@313	@314	2	4	

주소	...	313	314	315	316	317	...
데이터		10	'abc'	1	5		

```
var a;      var b = 'abc';
a = 10;    b = false;
```

변수명	a	b <span>1</span>	
주소	@313	@314 <span>2</span>	

주소	...	313	314	315	316	317	...
데이터		10	false <span>3</span>				

```

var a;           var b = 'abc';   var c = b;
a = 10;         b = false;

```

변수명	a	b	4	c	3	7
주소	@313	@314	5	@315	2	8

주소	...	313	314	315	316	317	...
데이터		10	false	6	false	1	9

```

var a;      var b = 'abc';  var c = b;
a = 10;    b = false;   var d = a;

```

변수명	a	b	c	d
주소	@313	@314	@315	@316
데이터	10	false	abc	abc

주소	...	313	314	315	316	317	...
데이터		10	false	abc	abc		

```

var a;      var b = 'abc';  var c = b;
a = 10;    b = false;   var d = a;

```

b ≡ c

변수명	a	b	c	d
주소	@313	@314	@315	@316
데이터	10	false	false	10

주소	...	313	314	315	316	317	...
데이터		10	false	false	10		

```

var a;      var b = 'abc';  var c = b;
a = 10;    b = false;   var d = a;

```

$b \equiv c$   
 $a \equiv d$

변수명	a	b	c	d
주소	@313	@314	@315	@316
데이터	10	false	'abc'	10

주소	...	313	314	315	316	317	...
데이터		10	false	false	10		

```
var obj = {
  a: 1,
  b: 'b'
};
```

변수명	obj	3
주소	@413	2

주소	...	413	414	...	1011	1012	1013	...
데이터		1 7			{ a: @1012, b: @1013 }	1	'b'	

```
var obj = {  
    a: 1,  
    b: 'b'  
};
```

```
var obj2 = obj;
```

변수명	obj	obj2		
주소	@413	@414		
데이터	...	413	414	...

주소	...	413	414	...	1011	1012	1013	...
데이터		@1011	@1011		{ a: @1012, b: @1013 }	1	'b'	

```

var obj = {
  a: 1,
  b: 'b'
};

```

```

var obj2 = obj;

```

변수명	obj	obj2		
주소	@413	@414		
데이터				

주소	...	413	414	...	1011	1012	1013	...
데이터					{ a: @1012, b: @1013 }	10	'b'	

```

var obj = {
  a: 1,
  b: 'b'
};

```

obj  obj2

```
var obj2 = obj;
```

변수명	obj	obj2		
주소	@413	@414		
데이터	...	413	414	...

주소	...	413	414	...	1011	1012	1013	...
데이터		@1011	@1011		{ a: @1012, b: @1013 }	10	10	

```
var obj3 = {  
    a: [4, 5, 6]  
};
```

변수명	obj3	3
주소	@547	2

주소	...	547	...	1184	1185	...	1326	1327	1328	...
데이터		1 8		{ a: @1185 }	[ @1326, @1327, @1328 ]		4	5	5	

```
var obj3 = {  
    a: [4, 5, 6]  
};
```

```
obj3.a = 'new';
```

변수명	obj3			
주소	@547			

G.C  
대상

주소	...	547	...	1184	1185	...	1326	1327	1328	...
데이터		@1184		{ a: @1185 }	'new'		4	5	6	

## Primitive Type

값을 그대로 할당

Number  
String  
Boolean  
null  
undefined

## Reference Type — Object

값이 저장된 주소값을 할당 (참조)

Array  
Function  
RegExp

jsFlow 2017. 9. 9. gomugom

---

# 2. FUNCTION

**2-1.  
FUNCTION?**

함수 키워드      함수명      인수부

function sum(a, b) {

    return a + b;

}

반환 명령

반환할 값

} 함수본문

선언O

실행X

```
function sum(a, b) {  
    return a + b;  
}  
console.dir(sum);
```

i *function sum(a, b)*  
arguments: null  
caller: null  
length: 2  
name: "sum"  
prototype: Object  
\_\_proto\_\_: *function ()*  
[[FunctionLocation]]:  
[[Scopes]]: Scopes [1]

실행시 인수의 집합 (배열X)

호출한 함수 (전역: null)

정의된 인수의 개수

함수 이름

(이하 추후 설명)

i *function sum(a, b)*  
arguments: null  
caller: null  
length: 2  
name: "sum"  
prototype: Object  
\_\_proto\_\_: *function ()*  
[[FunctionLocation]]:  
[[Scopes]]: Scopes[1]

함수 sum 선언 (정의)

```
function sum(a, b) {  
    return a + b;  
}
```

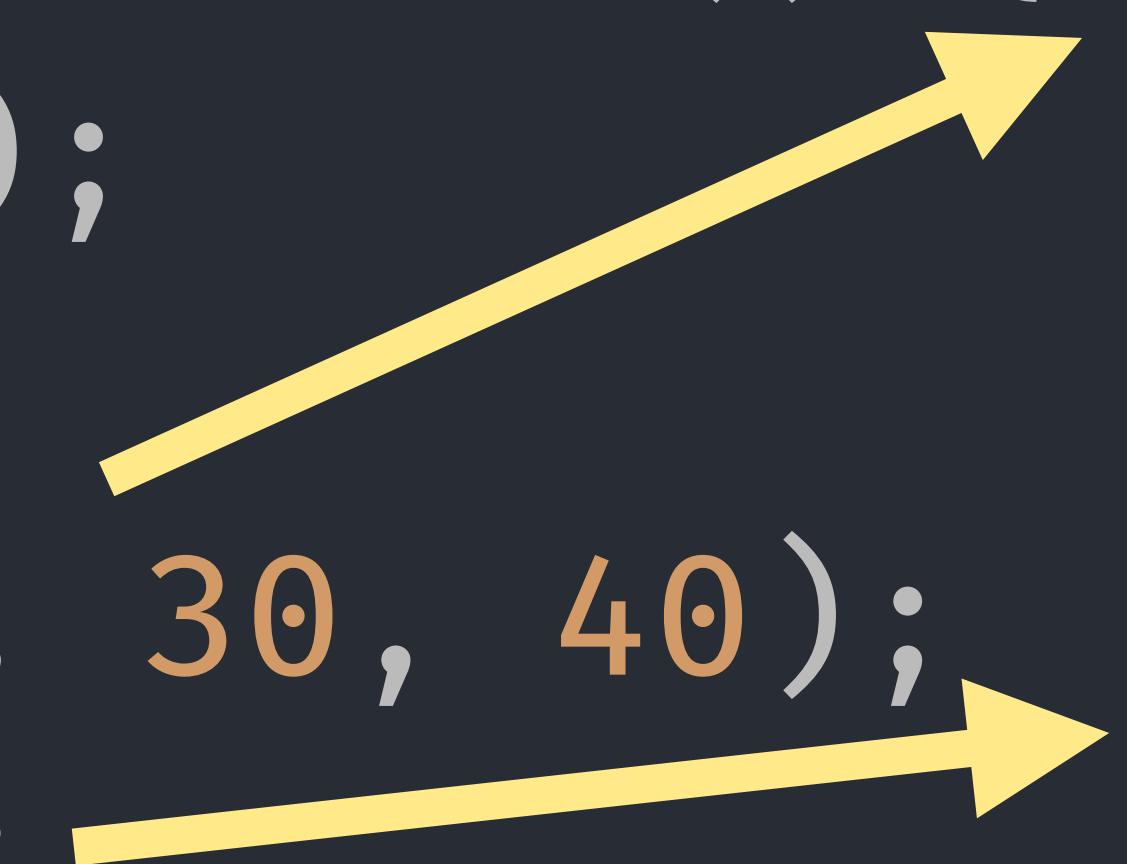
함수 sum 호출 (인수: 1, 2)

```
sum(1, 2);
```

함수 sum 호출 (인수: 2, 3)  
결과값을 변수 res에 할당.

```
var res = sum(2, 3);
```

```
// 2-1-2
function sum(a, b) {
  console.log(sum.caller);
  console.log(arguments);
  return a + b;
}
function callFunc() {
  sum(1, 2);
}
sum(10, 20, 30, 40);
callFunc();
```



```
null
▶ [10, 20, 30, 40, callee:
  function callFunc() {
    sum(1, 2);
  }
▶ [1, 2, callee: function,
```

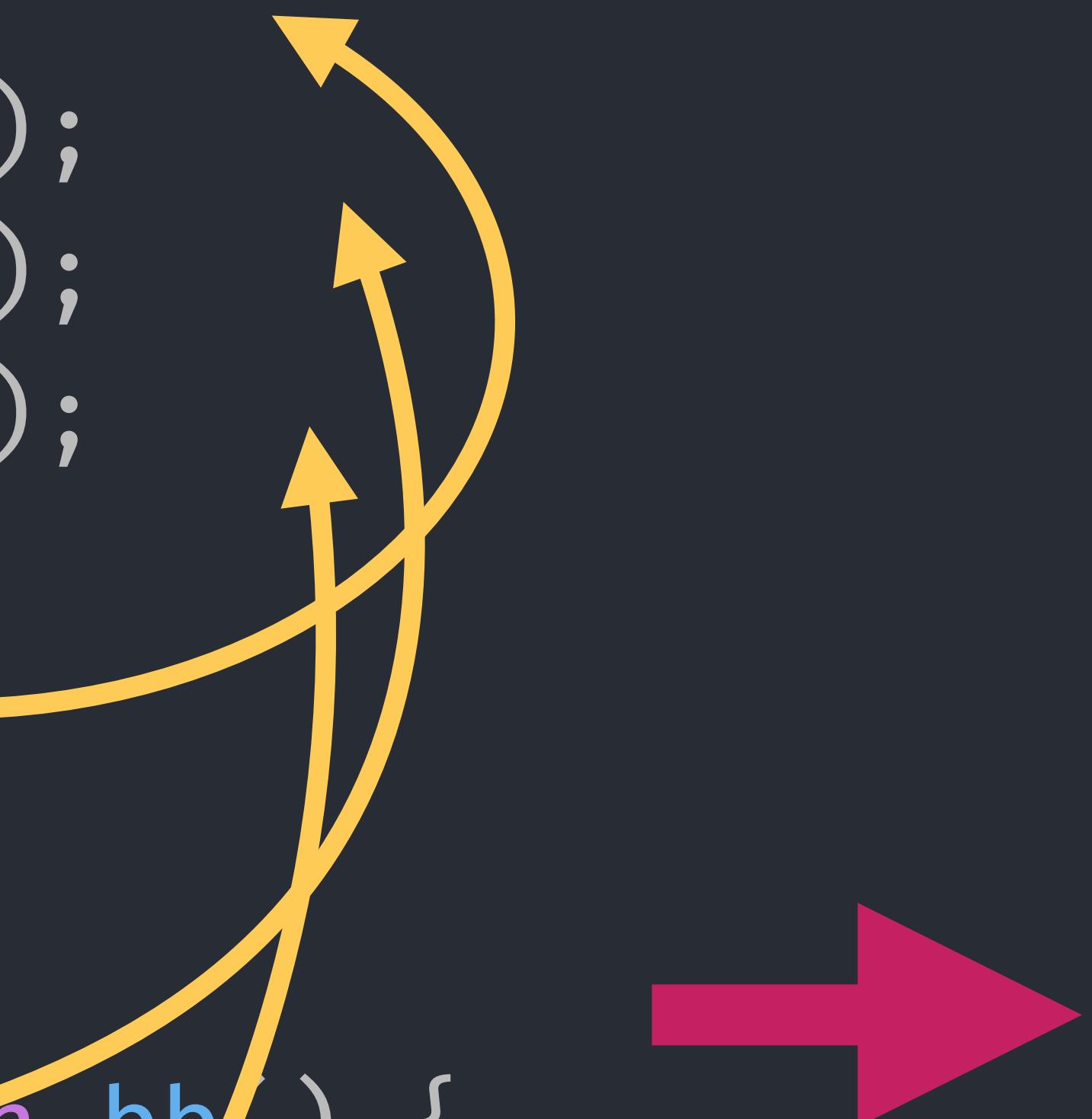
# 2-2. HOISTING

```
// 2-2-1
console.log(a());
console.log(b());
console.log(c());

function a() {
    return 'a';
}

var b = function bb() {
    return 'bb';
}

var c = function() {
    return 'c';
}
```



```
function a() {
    return 'a';
}

var b;
var c;

console.log(a());
console.log(b());
console.log(c());

b = function bb() {
    return 'bb';
}

c = function() {
    return 'c';
}
```

2-3.

# FUNCTION DECLARATION VS. FUNCTION EXPRESSION

함수선언문

function declaration

기명 함수표현식

named function expression

(익명) 함수표현식

(unnamed / anonymous)

function expression

```
// 2-3-1
function a() {
    return 'a';
}

var b = function bb() {
    return 'bb';
}

var c = function() {
    return 'c';
}
```

변수 c 선언

익명함수 선언

변수 c에 익명함수를 할당

```
var c = function() {  
    return 'c';  
}
```



```
// 2-3-2
```

```
function sum(a, b) {  
    return a + ' + ' + b + ' = ' + (a + b);
```

```
}
```

```
sum(1, 2);
```

```
/* ... 중략 ... */
```

```
function sum(a, b)  
    return a + b;
```

```
}
```

```
sum(3, 4);
```

```
var sum = function(a, b) {  
    return a + ' + ' + b + ' = ' + (a + b);
```

```
}
```

```
sum(1, 2);
```

```
{ /* ... 중략 ... */
```

```
var sum = function(a, b) {  
    return a + b;
```

```
}
```

# 2-4. FUNCTION SCOPE, EXECUTION CONTEXT

**스코프 유효범위 (변수)**  
SCOPE

**실행 컨텍스트 실행되는 코드덩어리**  
EXECUTION CONTEXT (추상적 개념)

스코프는  
정의될 때 결정된다!!

실행 컨텍스트는  
실행될 때 생성된다!!

실행 컨텍스트가  
생성될 때  
this 바인딩, 호이스팅  
등이 이뤄진다.

// 2-4-1

```
var a = 1;
function outer() {
    console.log(a);

    function inner() {
        console.log(a);
        var a = 3;
    }

    inner();
    console.log(a);
}
outer();
console.log(a);
```

## 0. 전역 실행컨텍스트 생성 [ GLOBAL ]

// 2-4-1

```
var a = 1;
function outer() {
    console.log(a);

    function inner() {
        console.log(a);
        var a = 3;
    }

    inner();
    console.log(a);
}
outer();
console.log(a);
```

## 0. 전역 실행컨텍스트 생성 [ GLOBAL ]

전역 컨텍스트

// 2-4-1

```
var a = 1;
function outer() {
    console.log(a);

    function inner() {
        console.log(a);
        var a = 3;
    }

    inner();
    console.log(a);
}
outer();
console.log(a);
```

## 0. 전역 실행컨텍스트 생성 [ GLOBAL ]

### 1. 변수 a 선언

전역 컨텍스트

// 2-4-1

```
var a = 1;
function outer() {
    console.log(a);

    function inner() {
        console.log(a);
        var a = 3;
    }

    inner();
    console.log(a);
}

outer();
console.log(a);
```

## 0. 전역 실행컨텍스트 생성 [ GLOBAL ]

1. 변수 a 선언

2. 함수 outer 선언 [ GLOBAL > outer ]

전역 컨텍스트

// 2-4-1

```
var a = 1;
function outer() {
    console.log(a);

    function inner() {
        console.log(a);
        var a = 3;
    }

    inner();
    console.log(a);
}
outer();
console.log(a);
```

## 0. 전역 실행컨텍스트 생성 [ GLOBAL ]

1. 변수 a 선언

2. 함수 outer 선언 [ GLOBAL > outer ]

3. 변수 a에 1 할당

전역 컨텍스트

// 2-4-1

```
var a = 1;
function outer() {
    console.log(a);

    function inner() {
        console.log(a);
        var a = 3;
    }

    inner();
    console.log(a);
}

outer();
console.log(a);
```

## 0. 전역 실행컨텍스트 생성 [ GLOBAL ]

1. 변수 a 선언

2. 함수 outer 선언 [ GLOBAL > outer ]

3. 변수 a에 1 할당

4. outer 함수 호출 -> outer 실행컨텍스트 생성

전역 컨텍스트

// 2-4-1

```
var a = 1;
function outer() {
    console.log(a);

    function inner() {
        console.log(a);
        var a = 3;
    }

    inner();
    console.log(a);
}

outer(); // 종료 대기중 (1)
console.log(a);
```

## 0. 전역 실행컨텍스트 생성 [ GLOBAL ]

1. 변수 a 선언

2. 함수 outer 선언 [ GLOBAL > outer ]

3. 변수 a에 1 할당

4. outer 함수 호출 -> outer 실행컨텍스트 생성

전역 컨텍스트

OUTER 컨텍스트

// 2-4-1

```
var a = 1;  
function outer() {  
    console.log(a);  
  
    function inner() {  
        console.log(a);  
        var a = 3;  
    }  
}
```

```
inner();
```

```
console.log(a);  
}
```

outer 함수 종료 대기중 (1)

```
console.log(a);
```

## 0. 전역 실행컨텍스트 생성 [ GLOBAL ]

1. 변수 a 선언

2. 함수 outer 선언 [ GLOBAL > outer ]

3. 변수 a에 1 할당

4. outer 함수 호출 -> outer 실행컨텍스트 생성

5. 함수 inner 선언 [ GLOBAL > outer > inner ]

전역 컨텍스트

OUTER 컨텍스트

// 2-4-1

```
var a = 1;
function outer() {
    console.log(a);

    function inner() {
        console.log(a);
        var a = 3;
    }

    inner();
    console.log(a);
}

outer(); // 종료 대기중 (1)
console.log(a);
```

## 0. 전역 실행컨텍스트 생성 [ GLOBAL ]

1. 변수 a 선언

2. 함수 outer 선언 [ GLOBAL > outer ]

3. 변수 a에 1 할당

4. outer 함수 호출 -> outer 실행컨텍스트 생성

5. 함수 inner 선언 [ GLOBAL > outer > inner ]

6. outer scope에서 a 탐색 -> global scope에서 a 탐색 -> 1 출력

전역 컨텍스트

OUTER 컨텍스트

// 2-4-1

```
var a = 1;
function outer() {
    console.log(a);

    function inner() {
        console.log(a);
        var a = 3;
    }

    inner();
}

console.log(a);
}

outer(); // 종료 대기중 (1)
console.log(a);
```

## 0. 전역 실행컨텍스트 생성 [ GLOBAL ]

1. 변수 a 선언

2. 함수 outer 선언 [ GLOBAL > outer ]

3. 변수 a에 1 할당

4. outer 함수 호출 -> outer 실행컨텍스트 생성

5. 함수 inner 선언 [ GLOBAL > outer > inner ]

6. outer scope에서 a 탐색 -> global scope에서 a 탐색 -> 1 출력

7. inner 함수 호출 -> inner 실행컨텍스트 생성

전역 컨텍스트

OUTER 컨텍스트

// 2-4-1

```
var a = 1;
function outer() {
    console.log(a);

    function inner() {
        console.log(a);
        var a = 3;

        console.log(a);
    }
}

outer();
console.log(a);
```

inner함수 종료 대기중 (2)

}

outer함수 종료 대기중 (1)

console.log(a);

## 0. 전역 실행컨텍스트 생성 [ GLOBAL ]

1. 변수 a 선언

2. 함수 outer 선언 [ GLOBAL > outer ]

3. 변수 a에 1 할당

4. outer 함수 호출 -> outer 실행컨텍스트 생성

5. 함수 inner 선언 [ GLOBAL > outer > inner ]

6. outer scope에서 a 탐색 -> global scope에서 a 탐색 -> 1 출력

7. inner 함수 호출 -> inner 실행컨텍스트 생성

전역 컨텍스트

OUTER 컨텍스트

INNER 컨텍스트

// 2-4-1

```
var a = 1;
function outer() {
    console.log(a);

    function inner() {
        console.log(a);
        var a = 3;

        console.log(a);
    }
    outer함수 종료 대기중 (1)
    console.log(a);
```

## 0. 전역 실행컨텍스트 생성 [ GLOBAL ]

1. 변수 a 선언

2. 함수 outer 선언 [ GLOBAL > outer ]

3. 변수 a에 1 할당

4. outer 함수 호출 -> outer 실행컨텍스트 생성

5. 함수 inner 선언 [ GLOBAL > outer > inner ]

6. outer scope에서 a 탐색 -> global scope에서 a 탐색 -> 1 출력

7. inner 함수 호출 -> inner 실행컨텍스트 생성

8. 변수 a 선언

전역 컨텍스트

OUTER 컨텍스트

INNER 컨텍스트

// 2-4-1

```
var a = 1;
function outer() {
    console.log(a);

    function inner() {
        console.log(a);
        var a = 3;
    }

    inner();
    console.log(a);
}

outer();
console.log(a);
```

inner함수 종료 대기중 (2)

}

outer함수 종료 대기중 (1)

console.log(a);

## 0. 전역 실행컨텍스트 생성 [ GLOBAL ]

1. 변수 a 선언

2. 함수 outer 선언 [ GLOBAL > outer ]

3. 변수 a에 1 할당

4. outer 함수 호출 -> outer 실행컨텍스트 생성

5. 함수 inner 선언 [ GLOBAL > outer > inner ]

6. outer scope에서 a 탐색 -> global scope에서 a 탐색 -> 1 출력

7. inner 함수 호출 -> inner 실행컨텍스트 생성

8. 변수 a 선언

9. inner scope에서 a 탐색 -> undefined 출력

전역 컨텍스트

OUTER 컨텍스트

INNER 컨텍스트

// 2-4-1

```
var a = 1;
function outer() {
    console.log(a);

    function inner() {
        console.log(a);
        var a = 3;
    }

    inner();
    console.log(a);
}

outer();
console.log(a);
```

## 0. 전역 실행컨텍스트 생성 [ GLOBAL ]

1. 변수 a 선언

2. 함수 outer 선언 [ GLOBAL > outer ]

3. 변수 a에 1 할당

4. outer 함수 호출 -> outer 실행컨텍스트 생성

5. 함수 inner 선언 [ GLOBAL > outer > inner ]

6. outer scope에서 a 탐색 -> global scope에서 a 탐색 -> 1 출력

7. inner 함수 호출 -> inner 실행컨텍스트 생성

8. 변수 a 선언

9. inner scope에서 a 탐색 -> undefined 출력

10. 변수 a에 3 할당

전역 컨텍스트

OUTER 컨텍스트  
INNER

// 2-4-1

```
var a = 1;
function outer() {
    console.log(a);

    function inner() {
        console.log(a);
        var a = 3;
    }

    inner();
    console.log(a);
}

outer();
console.log(a);
```

inner(함수, 종료 ! (2))

outer(함수, 종료 대기중 (1))

console.log(a);

## 0. 전역 실행컨텍스트 생성 [ GLOBAL ]

1. 변수 a 선언

2. 함수 outer 선언 [ GLOBAL > outer ]

3. 변수 a에 1 할당

4. outer 함수 호출 -> outer 실행컨텍스트 생성

5. 함수 inner 선언 [ GLOBAL > outer > inner ]

6. outer scope에서 a 탐색 -> global scope에서 a 탐색 -> 1 출력

7. inner 함수 호출 -> inner 실행컨텍스트 생성

8. 변수 a 선언

9. inner scope에서 a 탐색 -> undefined 출력

10. 변수 a에 3 할당

11. inner 실행컨텍스트 종료

전역 컨텍스트

OUTER 컨텍스트  
INNER

// 2-4-1

```
var a = 1;
function outer() {
    console.log(a);

    function inner() {
        console.log(a);
        var a = 3;
    }

    inner();
    console.log(a);
}

outer(); // a 출력 대기중 (1)
console.log(a);
```

## 0. 전역 실행컨텍스트 생성 [GLOBAL]

1. 변수 a 선언

2. 함수 outer 선언 [GLOBAL > outer]

3. 변수 a에 1 할당

4. outer 함수 호출 -> outer 실행컨텍스트 생성

5. 함수 inner 선언 [GLOBAL > outer > inner]

6. outer scope에서 a 탐색 -> global scope에서 a 탐색 -> 1 출력

7. inner 함수 호출 -> inner 실행컨텍스트 생성

8. 변수 a 선언

9. inner scope에서 a 탐색 -> undefined 출력

10. 변수 a에 3 할당

11. inner 실행컨텍스트 종료

12. outer scope에서 a 탐색 -> global scope에서 a 탐색 -> 1 출력

전역 컨텍스트

OUTER 컨텍스트  
INNER

// 2-4-1

```
var a = 1;
function outer() {
    console.log(a);

    function inner() {
        console.log(a);
        var a = 3;
    }

    inner();
    console.log(a);
}

outer() 함수 종료!(1)
console.log(a);
```

## 0. 전역 실행컨텍스트 생성 [GLOBAL]

1. 변수 a 선언

2. 함수 outer 선언 [GLOBAL > outer]

3. 변수 a에 1 할당

4. outer 함수 호출 -> outer 실행컨텍스트 생성

5. 함수 inner 선언 [GLOBAL > outer > inner]

6. outer scope에서 a 탐색 -> global scope에서 a 탐색 -> 1 출력

7. inner 함수 호출 -> inner 실행컨텍스트 생성

8. 변수 a 선언

9. inner scope에서 a 탐색 -> undefined 출력

10. 변수 a에 3 할당

11. inner 실행컨텍스트 종료

12. outer scope에서 a 탐색 -> global scope에서 a 탐색 -> 1 출력

13. outer 실행컨텍스트 종료

전역 컨텍스트

OUTER 컨텍스트

INNER 컨텍스트

// 2-4-1

```
var a = 1;
function outer() {
    console.log(a);

    function inner() {
        console.log(a);
        var a = 3;
    }

    inner();
    console.log(a);
}
outer();
console.log(a);
```

## 0. 전역 실행컨텍스트 생성 [GLOBAL]

4. outer 함수 호출 -> outer 실행컨텍스트 생성

5. 함수 inner 선언 [ GLOBAL > outer > inner ]

6. outer scope에서 a 탐색 -> global scope에서 a 탐색 -> 1 출력

7. inner 함수 호출 -> inner 실행컨텍스트 생성

OUTER 컨텍스트  
INNER

8. 변수 a 선언

9. inner scope에서 a 탐색 -> undefined 출력

10. 변수 a에 3 할당

11. inner 실행컨텍스트 종료

12. outer scope에서 a 탐색 -> global scope에서 a 탐색 -> 1 출력

13. outer 실행컨텍스트 종료

14. global scope에서 a 탐색 -> 1 출력

// 2-4-1

```
var a = 1;
function outer() {
    console.log(a);

    function inner() {
        console.log(a);
        var a = 3;
    }

    inner();
    console.log(a);
}
outer();
console.log(a);
```

## 0. 전역 실행컨텍스트 생성 [GLOBAL]

4. outer 함수 호출 -> outer 실행컨텍스트 생성

5. 함수 inner 선언 [ GLOBAL > outer > inner ]

6. outer scope에서 a 탐색 -> global scope에서 a 탐색 -> 1 출력

7. inner 함수 호출 -> inner 실행컨텍스트 생성

OUTER 컨텍스트  
INNER

8. 변수 a 선언

9. inner scope에서 a 탐색 -> undefined 출력

10. 변수 a에 3 할당

11. inner 실행컨텍스트 종료

12. outer scope에서 a 탐색 -> global scope에서 a 탐색 -> 1 출력

13. outer 실행컨텍스트 종료

14. global scope에서 a 탐색 -> 1 출력

15. 전역 실행컨텍스트 종료

# 2-5. METHOD

// 2-5-1

```
var obj = {  
  a: 1,  
  b: function bb() {  
    console.log(this);  
  },  
  c: function() {  
    console.log(this.a);  
  }  
};
```

```
obj.b();  
obj.c();
```

```
console.dir(obj.b);  
console.dir(obj.c);
```

## 0. 전역 실행컨텍스트 생성 [ GLOBAL ]

// 2-5-1

```
var obj = {  
    a: 1,  
    b: function bb() {  
        console.log(this);  
    },  
    c: function() {  
        console.log(this.a);  
    }  
};
```

```
obj.b();  
obj.c();
```

```
console.dir(obj.b);  
console.dir(obj.c);
```

## 0. 전역 실행컨텍스트 생성 [ GLOBAL ]

### 1. 변수 obj 선언

전역 컨텍스트

// 2-5-1

```
var obj = {
  a: 1,
  b: function bb() {
    console.log(this);
  },
  c: function() {
    console.log(this.a);
  }
};
```

## 0. 전역 실행컨텍스트 생성 [ GLOBAL ]

1. 변수 obj 선언

2. 객체 생성 / 변수 obj에 객체 주소값 할당

전역 컨텍스트

```
obj.b();
obj.c();
```

```
console.dir(obj.b);
console.dir(obj.c);
```

// 2-5-1

```
var obj = {  
  a: 1,  
  b: function bb() {  
    console.log(this);  
  },  
  c: function() {  
    console.log(this.a);  
  }  
};
```

```
obj.b();  
obj.c();
```

```
console.dir(obj.b);  
console.dir(obj.c);
```

## 0. 전역 실행컨텍스트 생성 [ GLOBAL ]

1. 변수 obj 선언

2. 객체 생성 / 변수 obj에 객체 주소값 할당

3. obj.b 메소드 호출 -> obj.b 실행컨텍스트 생성

전역 컨텍스트

// 2-5-1

```
var obj = {  
  a: 1,  
  b: function bb() {  
    console.log(this);  
  },  
  c: function() {  
    console.log(this.a);  
  }  
};
```

```
obj.b();  
obj.c();
```

```
console.dir(obj.b);  
console.dir(obj.c);
```

## 0. 전역 실행컨텍스트 생성 [ GLOBAL ]

1. 변수 obj 선언

2. 객체 생성 / 변수 obj에 객체 주소값 할당

3. obj.b 메소드 호출 -> obj.b 실행컨텍스트 생성

4. this에 obj 바인딩

전역 컨텍스트

o  
b  
j  
.b

// 2-5-1

```
var obj = {  
  a: 1,  
  b: function bb() {  
    console.log(this);  
  },  
  c: function() {  
    console.log(this.a);  
  }  
};
```

```
obj.b();  
obj.c();
```

```
console.dir(obj.b);  
console.dir(obj.c);
```

## 0. 전역 실행컨텍스트 생성 [ GLOBAL ]

1. 변수 obj 선언

2. 객체 생성 / 변수 obj에 객체 주소값 할당

3. obj.b 메소드 호출 -> obj.b 실행컨텍스트 생성

4. this에 obj 바인딩

5. this 출력

전역 컨텍스트

obj.b

// 2-5-1

```
var obj = {  
  a: 1,  
  b: function bb() {  
    console.log(this);  
  },  
  c: function() {  
    console.log(this.a);  
  }  
};
```

```
obj.b();  
obj.c();
```

```
console.dir(obj.b);  
console.dir(obj.c);
```

## 0. 전역 실행컨텍스트 생성 [ GLOBAL ]

전역 컨텍스트

1. 변수 obj 선언
2. 객체 생성 / 변수 obj에 객체 주소값 할당
3. obj.b 메소드 호출 -> obj.b 실행컨텍스트 생성
4. this에 obj 바인딩
5. this 출력
6. obj.b 실행컨텍스트 종료

// 2-5-1

```
var obj = {  
  a: 1,  
  b: function bb() {  
    console.log(this);  
  },  
  c: function() {  
    console.log(this.a);  
  }  
};
```

```
obj.b();  
obj.c();
```

```
console.dir(obj.b);  
console.dir(obj.c);
```

## 0. 전역 실행컨텍스트 생성 [ GLOBAL ]

1. 변수 obj 선언

2. 객체 생성 / 변수 obj에 객체 주소값 할당

3. obj.b 메소드 호출 -> obj.b 실행컨텍스트 생성

4. this에 obj 바인딩

5. this 출력

6. obj.b 실행컨텍스트 종료

( ... 중략 ... )

7. 전역 실행컨텍스트 종료

전역 컨텍스트  
obj.b

# 2-6. CALLBACK FUNCTION

call / back

something

will call this function back

somehow sometime.

제어권을 넘겨준다.

맡긴다.

## 주기함수 호출

## 인자1: 콜백함수

```
setInterval(function () {  
    console.log('1초마다 실행될 겁니다.');//  
}, 1000);
```

인자2:  
주기 (ms)

```
var cb = function() {  
    console.log('1초마다 실행될 겁니다.');//  
};
```

```
setInterval(cb, 1000);
```

```
setInterval( callback, milliseconds )
```

```
// 2-6-2  
var arr = [1, 2, 3, 4, 5];  
var entries = [];  
arr.forEach(function(v, i) {  
    entries.push([i, v, this[i]]);  
}, [10, 20, 30, 40, 50]);  
console.log(entries);
```

```
[ [0, 1, 10], [1, 2, 20], [2, 3, 30], [3, 4, 40], [4, 5, 50] ]
```

// 2-6-2

```
var arr = [1, 2, 3, 4, 5];
```

```
var entries = [];
```

인자1: 콜백함수

```
arr.forEach(function(v, i) {
```

```
    entries.push([i, v, this[i]]);
```

```
}, [10, 20, 30, 40, 50]);
```

```
con
```

인자2: this로 인식할 대상  
(생략가능)

# Array.prototype.forEach()

현재 문서 내

**forEach()** 메소드는 배열 요소마다 한 번씩 제공한 함수를 실행합니다.

## 구문

```
arr.forEach(callback[, thisArg])
```

반환값

`undefined`.

매개변수

**callback**

각 요소에 대해 실행할 함수, 인수 셋을 취하는:

**currentValue**

배열에서 현재 처리 중인 요소.

**index**

배열에서 현재 처리 중인 요소의 인덱스.

**array**

`forEach()`가 적용되고 있는 배열.

**thisArg**

선택 사항. `callback`을 실행할 때 `this`로서 사용하는 값.

[https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global\\_Objects/Array/forEach](https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/Array/forEach)

// 2-6-3

```
Array.prototype.forEach = function(callback, thisArg) {  
    var self = thisArg || this;  
    for(var i = 0; i < this.length; i++) {  
        callback.call(self, this[i], i, this);  
    }  
}
```

```
// 2-6-4  
document.body.innerHTML = '<div id="a">abc</div>';  
function cbFunc(x) {  
    console.log(this, x);  
}
```

```
document.getElementById('a')  
    .addEventListener('click', cbFunc);  
  
$('#a').on('click', cbFunc);
```

```
<div id="a">abc</div>
```

```
▶ MouseEvent {isTrusted: true, screenX: 11,
```

```
target.addEventListener(type, listener[, useCapture]);
```

### type

등록할 event type 을 나타내는 문자열

### listener

특정 타입의 이벤트가 발생할 때 알림을 받을 객체. 반드시 **EventListener** 인터페이스를 수행하는 객체이거나, Javascript function 이어야 합니다.

### useCapture Optional

만약 true라면, useCapture 는 사용자가 capture를 초기화하길 원한다는 것을 나타냅니다. capture를 초기화 한 절차는 이벤트는 DOM tree에서 하위의 어떤 EventTarget에 전달되기 전에 드로우 리시너에게 먼저 전달됩니다.

<https://developer.mozilla.org/ko/docs/Web/API/EventTarget/addEventListener>

The **addEventListener (type, callback, options)** method, when invoked, must run these steps:

<https://dom.spec.whatwg.org/#dom-eventtarget-addeventlistener>

# 콜백함수의 특징

- ▶ 다른 함수(A)의 매개변수로 콜백함수(B)를 전달하면, A가 B의 제어권을 갖게 된다.
- ▶ 특별한 요청(bind)이 없는 한 A에 미리 정해진 방식에 따라 B를 호출한다.
- ▶ 미리 정해진 방식이란 `this`에 무엇을 바인딩할지, 매개변수에는 어떤 값들을 지정할지, 어떤 타이밍에 콜백을 호출할지 등이다.

주의!

콜백은 '함수'다.

```
// 2-6-5  
var obj = {  
    vals: [1, 2, 3],  
    logValues: function(v, i) {  
        if(this.vals) {  
            console.log(this.vals, v, i);  
        } else {  
            console.log(this, v, i);  
        }  
    }  
};
```

```
▶ {_vals: Array(3), logValues: f} 1 2
```

```
▶ Window {stop: f, open: f, alert: f, confirm: f, prompt: f, ...} 1 0  
▶ Window {stop: f, open: f, alert: f, confirm: f, prompt: f, ...} 2 1  
▶ Window {stop: f, open: f, alert: f, confirm: f, prompt: f, ...} 3 2  
▶ Window {stop: f, open: f, alert: f, confirm: f, prompt: f, ...} 4 3  
▶ Window {stop: f, open: f, alert: f, confirm: f, prompt: f, ...} 5 4
```

메소드로 호출

```
obj.logValues(1, 2);
```

콜백함수로 전달

```
arr.forEach(obj.logValues);
```

# 2-7. CONSTRUCTOR

// 2-7-1

```
function Person(name, age) {  
    this.name = name;  
    this.age = age;  
}  
var gomugom = new Person('고무', 20);
```

생성자 함수

Person의  
인스턴스

instance : 어떤 집합에 속하는 개별적인 요소  
어떤 등급(Class)에 속하는 각 객체

gomugom

▶ *Person {name: "고무", age: 20}*

```
// 2-7-2  
function Person(name, age) {  
    this.name = name;  
    this.age = age;  
    return name + ', ' + age;  
}  
var gomu1 = new Person('고무', 10);  
var gomu2 = Person('곰', 20);
```

Person의 인스턴스

Person 함수의 실행 결과

jsFlow 2017. 9. 9. gomugom

---

3. THIS

### 3-1. 전역공간에서

```
/* 브라우저 콘솔에서 */
console.log(this);
```

```
▼ Window ⓘ
  ► $: function (e,t)
  ► $Add: Object
  ► $App: Object
  ▼ $Edit: Object
    ► ini: function ()
    ► loadEdit: function ()
    ► onedit: function ()
    ► __proto__: Object
  ► $Historys: Object
  ► $Notes: Object
  ► $Setting: Object
  ► $Slide: function (t,e,n,i)
  ► $Todos: Object
  ► $addButton: Object
```

```
/* node.js에서 */
console.log(this);
```

```
{ DTRACE_NET_SERVER_CONNECTION: [Function],
  DTRACE_NET_STREAM_END: [Function],
  DTRACE_HTTP_SERVER_REQUEST: [Function],
  DTRACE_HTTP_SERVER_RESPONSE: [Function],
  DTRACE_HTTP_CLIENT_REQUEST: [Function],
  DTRACE_HTTP_CLIENT_RESPONSE: [Function],
  global: [Circular],
  process:
    process {
      title: 'node',
      version: 'v7.9.0',
      moduleLoadList:
        [ 'Binding contextify',
          'Binding natives',
          'NativeModule events' ] }
```

### 3-2. 함수 내부에서

```
// 3-2-1.

function a() {
    console.log(this);
}
a();

function b() {
    function c() {
        console.log(this);
    }
    c();
}
b();

var d = {
    e: function() {
        function f() {
            console.log(this);
        }
        f();
    }
}
d.e();
```

#### ▼ Window i

- ▶ \$: *function (e,t)*
- ▶ \$Add: Object
- ▶ \$App: Object
- ▶ \$Edit: Object
  - ▶ ini: *function ()*
  - ▶ loadEdit: *function ()*
  - ▶ onedit: *function ()*
  - ▶ \_\_proto\_\_: Object
- ▶ \$Historys: Object
- ▶ \$Notes: Object
- ▶ \$Setting: Object
- ▶ \$Slide: *function (t,e,n,i)*
- ▶ \$Todos: Object
- ▶ \$addButton: Object
- ▶ \$addIcon: Object
- ▶ \$bookmarks: Object

### 3-3. 메소드 호출시

```
// 3-3-1.
```

```
var a = {  
  b: function() {  
    console.log(this);  
  }  
}
```

```
a.b();
```

```
▼ Object {b: function}  
  ► b: function ()  
  ► __proto__: Object
```

```
var a = {  
  b: {  
    c: function() {  
      console.log(this);  
    }  
  }  
};
```

```
a.b.c();
```

```
▼ Object {c: function}  
  ► c: function ()  
  ► __proto__: Object
```

### 3-3-1. 내부함수에서의 우회법

```
var a = 10;  
var obj = {  
  a: 20,  
  b: function() {  
    console.log(this.a); 20  
  }  
};  
obj.b();
```

20

여기서도 **this**를 쓸 수는 없을까?

### 3-3-1. 내부함수에서의 우회법

---

```
var a = 10;  
var obj = {  
  a: 20,  
  b: function() {  
    console.log(this.a);  
  
    function c() {  
      console.log(this.a);  
    }  
    c();  
  }  
}  
obj.b();
```

```
var a = 10;  
var obj = {  
  a: 20,  
  b: function() {  
    var self = this;  
    console.log(this.a);  
  
    function c() {  
      console.log(self.a);  
    }  
    c();  
  }  
}  
obj.b();
```

함수는  
(전역객체의)

메소드다! (라고 생각하자)

### 3-4. 콜백함수에서 - 명시적인 this 바인딩

```
▶ Object {c: "eee"} 1 2 3  
▶ Object {c: "eee"} 1 2 3  
▶ Object {c: "eee"} 1 2 3  
▶ Object {c: "eee"} 1 2 3
```

func.call(thisArg[, arg1[, arg2[, ... ]]])

func.apply(thisArg, [argsArray])

func.bind(thisArg[, arg1[, arg2[, ... ]]])

[https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global\\_Objects/Function/call](https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/Function/call)  
[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Function/apply](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/apply)  
[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Function/bind](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/bind)

```
// 3-4-1.  
function a(x, y, z) {  
    console.log(this, x, y, z);  
}  
var b = {  
    c: 'eee'  
};
```

a.call(b, 1, 2, 3);

a.apply(b, [1, 2, 3]);

var c = a.bind(b);  
c(1, 2, 3);

var d = a.bind(b, 1, 2);  
d(3);

### 3-4. 콜백함수에서 - 명시적인 this 바인딩

```
// 3-4-1.  
function a(x, y, z) {  
    console.log(this, x, y, z);  
}  
var b = {  
    c: 'eee'  
};
```

func.call(thisArg[, arg1[, arg2[, ... ]]])      즉시 호출      2, 3);

func.apply(thisArg, [argsArray])      즉시 호출      , 2, 3]);

func.bind(thisArg[, arg1[, arg2[, ... ]]])      새로운 함수 생성 (currying)  
c(1, 2, 3);

[https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global\\_Objects/Function/call](https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/Function/call)  
[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Function/apply](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/apply)  
[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Function/bind](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/bind)

```
var d = a.bind(b, 1, 2);  
d(3);
```

### 3-4. 콜백함수에서

```
// 3-4-2.  
var callback = function() {  
    console.dir(this);  
};  
var obj = {  
    a: 1,  
    b: function(cb) {  
        cb();  
    }  
};  
obj.b(callback);
```

▶ Window

### 3-4. 콜백함수에서

```
var callback = function() {
    console.dir(this);
};

var obj = {
    a: 1,
    b: function(cb) {
        cb && cb.call(this);
    }
};

obj.b(callback);
```

```
▼ Object i
  a: 1
  ► b: function (cb)
  ► __proto__: Object
```

### 3-4. 콜백함수에서

---

```
// 3-4-3.  
var callback = function() {  
    console.dir(this);  
};  
var obj = {  
    a: 1  
};  
setTimeout(callback, 100);
```

### 3-4. 콜백함수에서

---

```
// 3-4-3.  
var callback = function() {  
    console.dir(this);  
};  
var obj = {  
    a: 1  
};  
setTimeout(callback, 100);
```

▶ Window

### 3-4. 콜백함수에서

```
// 3-4-3.  
var callback = function() {  
    console.dir(this);  
};  
var obj = {  
    a: 1  
};  
setTimeout(callback.bind(obj), 100);
```

```
▼ Object i  
  a: 1  
► __proto__: Object
```

```
// 3-4-4.  
document.body.innerHTML += '<div id="a">클릭하세요</div>';
```

```
document.getElementById('a')  
.addEventListener('click', function() {  
    console.dir(this);  
});
```

```
▼ div#a ⓘ  
  accessKey: ""  
  align: ""  
  assignedSlot: null  
► attributes: NamedNodeMap  
  baseURI: "chrome-extension://c  
  childElementCount: 0  
► childNodes: NodeList(1)
```

# 정리

- ▶ 기본적으로는 함수의 this와 같다.
- ▶ 제어권을 가진 함수가 callback의 this를 명시한 경우 그에 따른다.
- ▶ 개발자가 this를 바인딩한 채로 callback을 넘기면 그에 따른다.

### 3-5. 생성자 함수에서

---

```
// 3-5-1.  
function Person(n, a) {  
    this.name = n;  
    this.age = a;  
}  
var gomugom = new Person('고무곰', 30);  
console.log(gomugom);
```

전역공간에서

window / global

함수 내부에서

window / global

메소드 호출시

메소드 호출 주체 (메소드명 앞)

callback에서

기본적으로는 함수내부에서와 동일

생성자함수에서

인스턴스

jsFlow 2017. 9. 9. gomugom

---

# 4. CLOSURE

# 4-1. CLOSURE?

A *closure* is the combination of a function and the lexical environment within which that function was declared.

선언 당시의 환경에 대한 정보를 담는 객체  
(구성 환경)

함수와  
“그 함수가 선언될 당시의 환경정보”  
사이의 조합

---

A *closure* is the combination of a function and the lexical environment within which that function was declared.

---

함수 내부에서 생성한 데이터와  
그 **유효범위**로 인해 발생하는  
특수한 **현상 / 상태**

# closure

닫혀있음 . 폐쇄성 . 완결성

# return

외부에 정보를 제공할 수 있는  
유일한 수단

return function  
scope 및 lexical environment는  
변하지 않는다.

return function

최초 선언시의 정보를 유지!

접근 권한 제어

지역변수 보호

데이터 보존 및 활용

// 4-1-1

```
function a() {  
    var x = 1;  
    function b() {  
        console.log(x);  
    }  
    b();  
}  
a();  
console.log(x);
```



접근 권한 제어

지역변수 보호

데이터 보존 및 활용

```
function a() {  
    var x = 1;  
    return function b() {  
        console.log(x);  
    };  
}  
  
var c = a();  
c();
```

접근 권한 제어

지역변수 보호

데이터 보존 및 활용

```
function a() {  
    var _x = 1;  
    return {  
        get x() { return _x; },  
        set x(v) { _x = v; }  
    };  
}  
  
var c = a();  
c.x = 10;
```

접근 권한 제어

지역변수 보호

데이터 보존 및 활용

```
function a() {  
    var _x = 1;  
    return {  
        get x() { return _x; },  
        set x(v) { _x = v; }  
    };  
}  
  
var c = a();  
c.x = 10;
```

## 0. 전역 실행컨텍스트 생성 [ GLOBAL ]

```
// 4-1-1

function setName(name) {
    return function() {
        return name;
    }
}

var sayMyName = setName('고무곰');
sayMyName();
```

## 0. 전역 실행컨텍스트 생성 [ GLOBAL ]

### 1. 함수 setName 선언 [ GLOBAL > setName ]

```
// 4-1-1  
  
function setName(name) {  
    return function() {  
        return name;  
    }  
}  
  
var sayMyName = setName('고무곰');  
sayMyName();
```

전역 컨텍스트

## 0. 전역 실행컨텍스트 생성 [ GLOBAL ]

1. 함수 setName 선언 [ GLOBAL > setName ]

2. 변수 sayMyName 선언

```
// 4-1-1

function setName(name) {
    return function() {
        return name;
    }
}

var sayMyName = setName('고무곰');
sayMyName();
```

전역 컨텍스트

## 0. 전역 실행컨텍스트 생성 [ GLOBAL ]

```
// 4-1-1  
function setName(name) {  
    return function() {  
        return name;  
    }  
}  
  
var sayMyName = setName('고무곰');  
sayMyName();
```

1. 함수 `setName` 선언 [ GLOBAL > `setName` ]

2. 변수 `sayMyName` 선언

3. `setName('고무곰')` 호출 -> `setName` 실행컨텍스트 생성

전역 컨텍스트

## 0. 전역 실행컨텍스트 생성 [ GLOBAL ]

```
// 4-1-1  
function setName(name) {  
    return function() {  
        return name;  
    }  
}  
  
var sayMyName = setName('고무곰');  
sayMyName();
```

전역 컨텍스트

1. 함수 setName 선언 [ GLOBAL > setName ]

2. 변수 sayMyName 선언

3. setName('고무곰') 호출 -> setName 실행컨텍스트 생성

4. 지역변수 name 선언 및 '고무곰' 할당

setName  
컨텍스트

## 0. 전역 실행컨텍스트 생성 [ GLOBAL ]

```
// 4-1-1  
function setName(name) {  
    return function() {  
        return name;  
    }  
}  
  
var sayMyName = setName('고무곰');  
sayMyName();
```

전역 컨텍스트

1. 함수 setName 선언 [ GLOBAL > setName ]

2. 변수 sayMyName 선언

3. setName('고무곰') 호출 -> setName 실행컨텍스트 생성

4. 지역변수 name 선언 및 '고무곰' 할당

5. 익명함수 선언 [ GLOBAL > setName > unnamed ]

setName  
컨텍스트

## 0. 전역 실행컨텍스트 생성 [ GLOBAL ]

```
// 4-1-1  
function setName(name) {  
    return function() {  
        return name;  
    }  
}  
  
var sayMyName = setName('고무곰');  
sayMyName();
```

전역 컨텍스트

1. 함수 setName 선언 [ GLOBAL > setName ]

2. 변수 sayMyName 선언

3. setName('고무곰') 호출 -> setName 실행컨텍스트 생성

4. 지역변수 name 선언 및 '고무곰' 할당

5. 익명함수 선언 [ GLOBAL > setName > unnamed ]

6. 익명함수 반환

setName  
컨텍스트

## 0. 전역 실행컨텍스트 생성 [ GLOBAL ]

```
// 4-1-1  
function setName(name) {  
    return function() {  
        return name;  
    }  
}  
  
var sayMyName = setName('고무곰');  
sayMyName();
```

전역 컨텍스트

1. 함수 setName 선언 [ GLOBAL > setName ]

2. 변수 sayMyName 선언

3. setName('고무곰') 호출 -> setName 실행컨텍스트 생성

setName  
컨텍스트

4. 지역변수 name 선언 및 '고무곰' 할당

5. 익명함수 선언 [ GLOBAL > setName > unnamed ]

6. 익명함수 반환

7. setName 실행컨텍스트 종료

## 0. 전역 실행컨텍스트 생성 [ GLOBAL ]

```
// 4-1-1  
function setName(name) {  
    return function() {  
        return name;  
    }  
}  
  
var sayMyName = setName('고무곰');  
sayMyName();
```

전역 컨텍스트

1. 함수 setName 선언 [ GLOBAL > setName ]

2. 변수 sayMyName 선언

3. setName('고무곰') 호출 -> setName 실행컨텍스트 생성

4. 지역변수 name 선언 및 '고무곰' 할당

5. 익명함수 선언 [ GLOBAL > setName > unnamed ]

6. 익명함수 반환

7. setName 실행컨텍스트 종료

8. 변수 sayMyName에 반환된 함수를 할당

setName  
컨텍스트

## 0. 전역 실행컨텍스트 생성 [ GLOBAL ]

```
// 4-1-1  
function setName(name) {  
    return function() {  
        return name;  
    }  
}  
  
var sayMyName = setName('고무곰');  
sayMyName();
```

전역 컨텍스트

1. 함수 `setName` 선언 [ GLOBAL > `setName` ]
2. 변수 `sayMyName` 선언
3. `setName('고무곰')` 호출 -> `setName` 실행컨텍스트 생성
4. 지역변수 `name` 선언 및 '고무곰' 할당
5. 익명함수 선언 [ GLOBAL > `setName` > unnamed ]
6. 익명함수 반환
7. `setName` 실행컨텍스트 종료
8. 변수 `sayMyName`에 반환된 함수를 할당
9. `sayMyName` 호출 -> `sayMyName` 실행컨텍스트 생성

setName  
컨텍스트

## 0. 전역 실행컨텍스트 생성 [ GLOBAL ]

```
// 4-1-1  
function setName(name) {  
    return function() {  
        return name;  
    }  
}  
  
var sayMyName = setName('고무곰');  
sayMyName();
```

전역 컨텍스트

1. 함수 setName 선언 [ GLOBAL > setName ]
2. 변수 sayMyName 선언
3. setName('고무곰') 호출 -> setName 실행컨텍스트 생성
4. 지역변수 name 선언 및 '고무곰' 할당
5. 익명함수 선언 [ GLOBAL > setName > unnamed ]
6. 익명함수 반환
7. setName 실행컨텍스트 종료
8. 변수 sayMyName에 반환된 함수를 할당
9. sayMyName 호출 -> sayMyName 실행컨텍스트 생성
10. unnamed scope에서 name 탐색 -> setName에서 name 탐색 -> '고무곰' 반환

sayMyName  
컨텍스트

## 0. 전역 실행컨텍스트 생성 [ GLOBAL ]

```
// 4-1-1  
function setName(name) {  
    return function() {  
        return name;  
    }  
}  
  
var sayMyName = setName('고무곰');  
sayMyName();
```

전역 컨텍스트

1. 함수 setName 선언 [ GLOBAL > setName ]
2. 변수 sayMyName 선언
3. setName('고무곰') 호출 -> setName 실행컨텍스트 생성
4. 지역변수 name 선언 및 '고무곰' 할당
5. 익명함수 선언 [ GLOBAL > setName > unnamed ]
6. 익명함수 반환
7. setName 실행컨텍스트 종료
8. 변수 sayMyName에 반환된 함수를 할당
9. sayMyName 호출 -> sayMyName 실행컨텍스트 생성
10. unnamed scope에서 name 탐색 -> setName에서 name 탐색 -> '고무곰' 반환
11. sayMyName 실행컨텍스트 종료

sayMyName  
컨텍스트

## 0. 전역 실행컨텍스트 생성 [ GLOBAL ]

```
// 4-1-1  
function setName(name) {  
    return function() {  
        return name;  
    }  
}  
  
var sayMyName = setName('고무곰');  
sayMyName();
```

전역 컨텍스트

1. 함수 setName 선언 [ GLOBAL > setName ]
2. 변수 sayMyName 선언
3. setName('고무곰') 호출 -> setName 실행컨텍스트 생성

setName  
컨텍스트

4. 지역변수 name 선언 및 '고무곰' 할당
5. 익명함수 선언 [ GLOBAL > setName > unnamed ]
6. 익명함수 반환

7. setName 실행컨텍스트 종료
8. 변수 sayMyName에 반환된 함수를 할당

9. sayMyName 호출 -> sayMyName 실행컨텍스트 생성

sayMyName  
컨텍스트

10. unnamed scope에서 name 탐색 -> setName에서 name 탐색 -> '고무곰' 반환

11. sayMyName 실행컨텍스트 종료

11. 전역 실행컨텍스트 종료

스코프는

정의될 때 결정된다!!

## setCounter 정의 [ GLOBAL > setCounter ]

```
// 4-1-2

function setCounter() {
    var count = 0;
    return function() {
        return ++count;
    }
}
var count = setCounter();
count();
```

**setCounter 정의 [ GLOBAL > setCounter ]**

```
// 4-1-2
function setCounter() {
    var count = 0;
    return function() {
        return ++count;
    }
}
var count = setCounter();
count();
```

**setCounter 실행**

```
// 4-1-2
function setCounter() {
    var count = 0;
    return function() {
        return ++count;
    }
}
var count = setCounter();
count();
```

setCounter 정의 [ GLOBAL > setCounter ]

setCounter 실행

setCounter 스코프에 count 변수 선언 및 0 할당

```
// 4-1-2
function setCounter() {
    var count = 0;
    return function() {
        return ++count;
    }
}
var count = setCounter();
count();
```

setCounter 정의 [ GLOBAL > setCounter ]

setCounter 실행

setCounter 스코프에 count 변수 선언 및 0 할당

익명함수 정의 및 반환 [ GLOBAL > setCounter > 익명 ]



```
// 4-1-2
function setCounter() {
    var count = 0;
    return function() {
        return ++count;
    }
}
var count = setCounter();
count();
```

setCounter 정의 [ GLOBAL > setCounter ]

setCounter 실행

setCounter 스코프에 count 변수 선언 및 0 할당

익명함수 정의 및 반환 [ GLOBAL > setCounter > 익명 ]

반환된 익명함수를 변수 count에 할당

```
// 4-1-2  
function setCounter() {  
    var count = 0;  
    return function() {  
        return ++count;  
    }  
}  
  
var count = setCounter();  
count();
```

**setCounter 정의 [ GLOBAL > setCounter ]**

**setCounter 실행**

**setCounter 스코프에 count 변수 선언 및 0 할당**

**익명함수 정의 및 반환 [ GLOBAL > setCounter > 익명 ]**

**반환된 익명함수를 변수 count에 할당**

**count 실행**

```
// 4-1-2
function setCounter() {
    var count = 0;
    return function() {
        return ++count;
    }
}
var count = setCounter();
count();
```

setCounter 정의 [ GLOBAL > setCounter ]

setCounter 실행

setCounter 스코프에 count 변수 선언 및 0 할당

익명함수 정의 및 반환 [ GLOBAL > setCounter > 익명 ]

반환된 익명함수를 변수 count에 할당

count 실행

익명함수 스코프에서 count 탐색

-> setCounter 스코프에서 count 탐색

-> count에 1을 증가시킨 값을 반환.

4-2.

# CLOSURE로 PRIVATE MEMBER 만들기

# - 자동차 게임 -

- ▶ 차량별로 연료량 및 연비는 랜덤
- ▶ 유저별로 차량 하나씩 고르면 게임 시작
- ▶ 각 유저는 자신의 턴에 주사위를 굴려  
랜덤하게 나온 숫자만큼 이동함
- ▶ 만약 연료가 부족하면 이동 불가
- ▶ 가장 멀리 간 사람이 승리

```
// 4-2-1
var car = {
  fuel: 10, // 연료 (l)
  power: 2, // 연비 (km / l)
  total: 0,
  run: function(km) {
    var wasteFuel = km / this.power;
    if(this.fuel < wasteFuel) {
      console.log('이동 불가');
      return;
    }
    this.fuel -= wasteFuel;
    this.total += km;
  }
};
```

```
car.power = 10;
car.fuel = 1000;
```

```
// 4-2-1
var car = {
  fuel: 10, // 연료 (l)
  power: 2, // 연비 (km / l)
  total: 0,
  run: function(km) {
    var wasteFuel = km / this.power;
    if(this.fuel < wasteFuel) {
      console.log('이동 불가');
      return;
    }
    this.fuel -= wasteFuel;
    this.total += km;
  }
};
```

car.power = 10;  
car.fuel = 1000;

```
var createCar = function(f, p) {
  var fuel = f;
  var power = p;
  var total = 0;
  return {
    run: function(km) {
      var wasteFuel = km / power;
      if(fuel < wasteFuel) {
        console.log('이동 불가');
        return;
      }
      fuel -= wasteFuel;
      total += km;
    }
  };
}
var car = createCar(10, 2);
```

1. 외부함수에서 지역변수 및 내부함수 등을 생성한다.

2. 외부에 노출시키고자 하는 멤버들로 구성된 객체를 return한다.

→ return한 객체에 포함되지 않은 멤버들은 private하다.

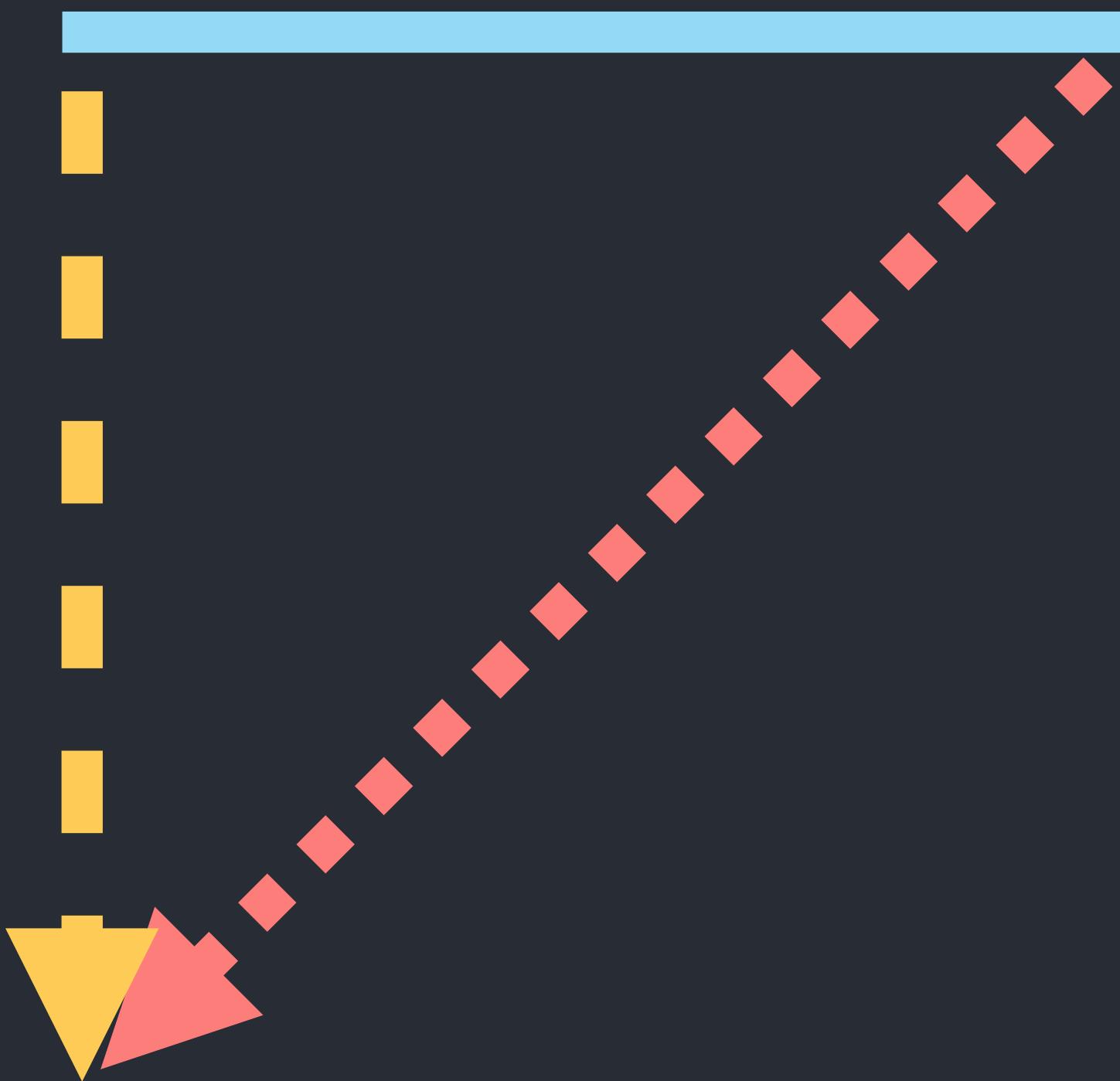
→ return한 객체에 포함된 멤버들은 public하다.

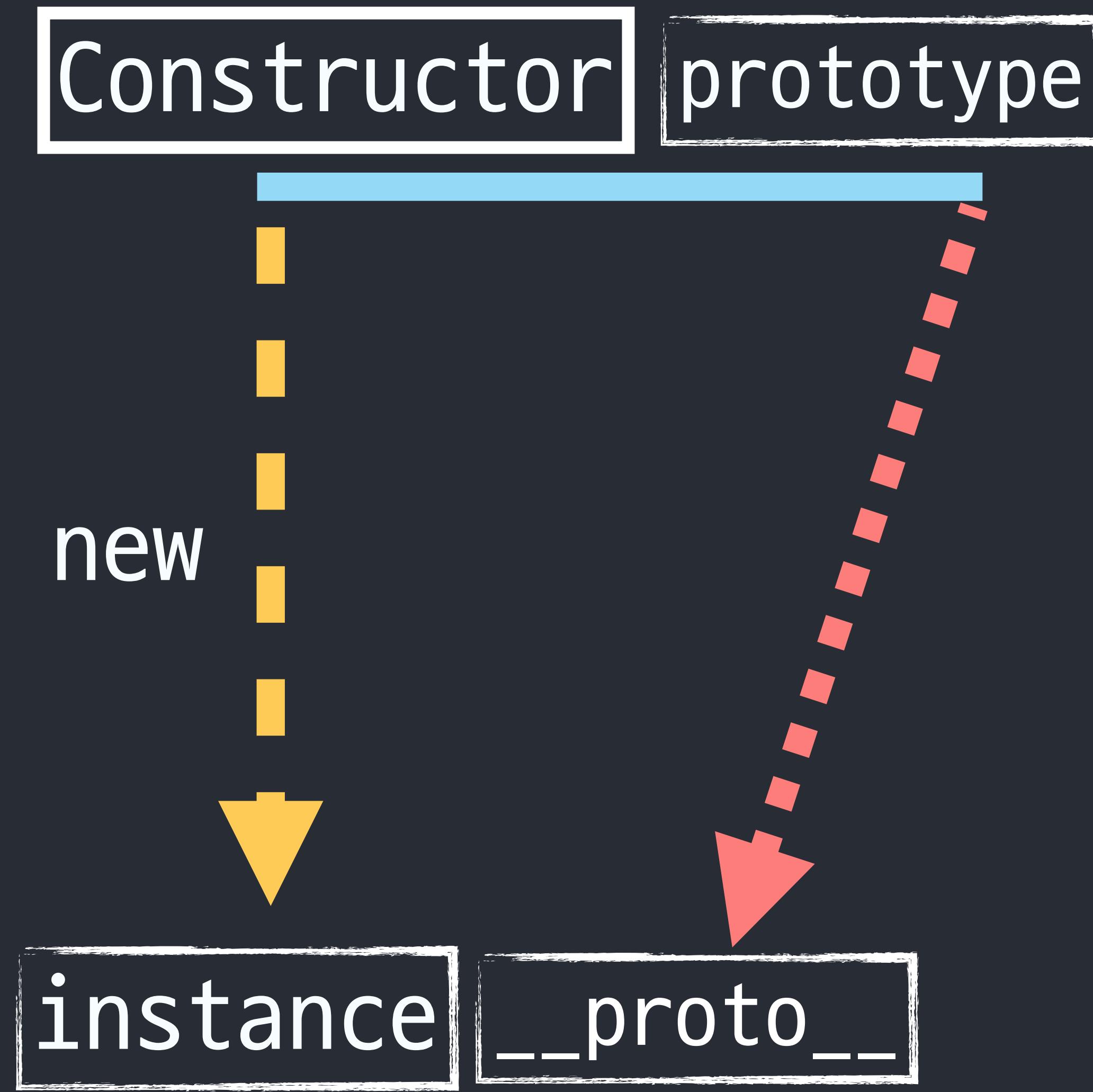
jsFlow 2017. 9. 9. gomugom

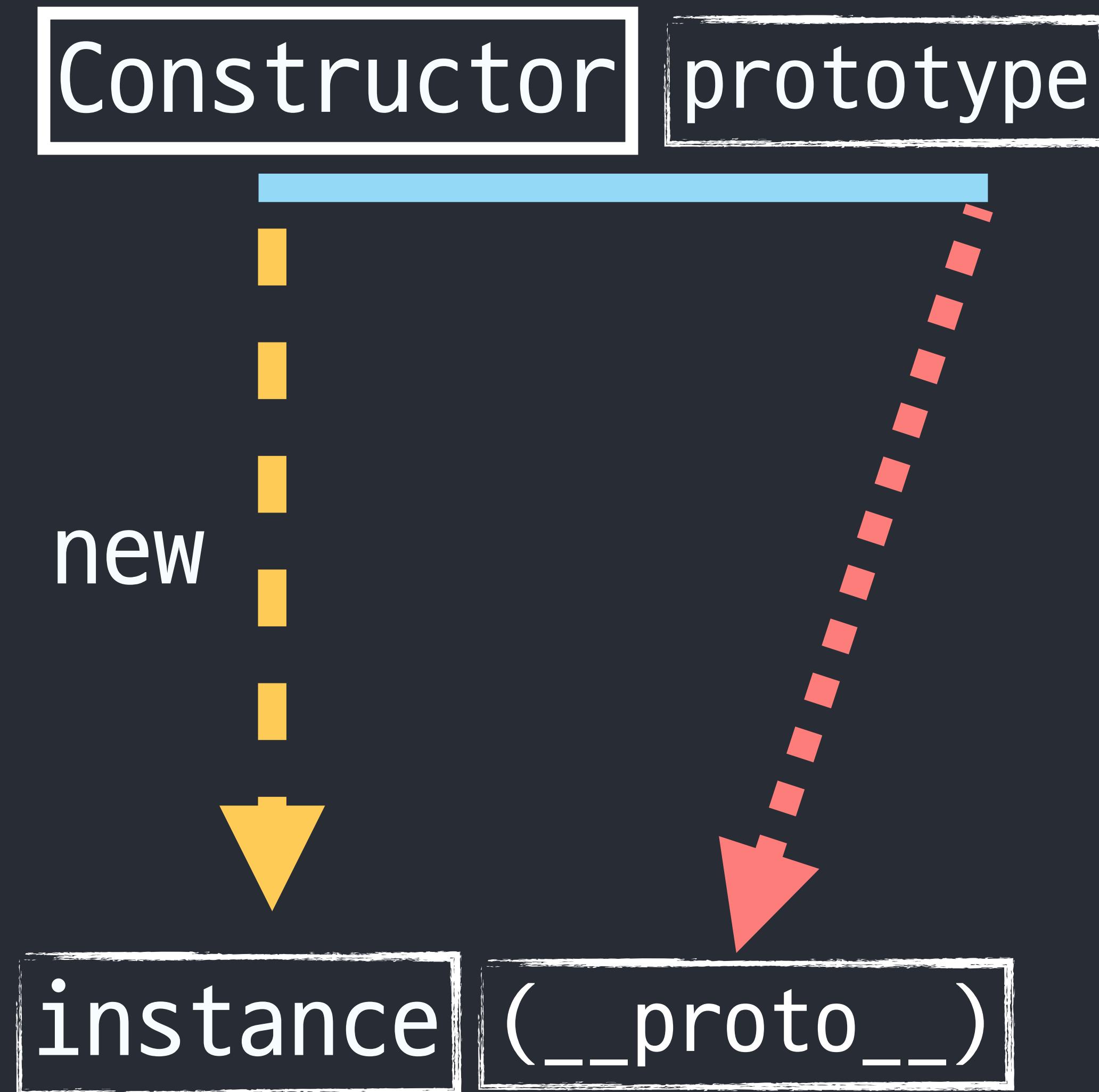
---

# 5. PROTOTYPE

5-1.  
prototype,  
constructor,  
proto



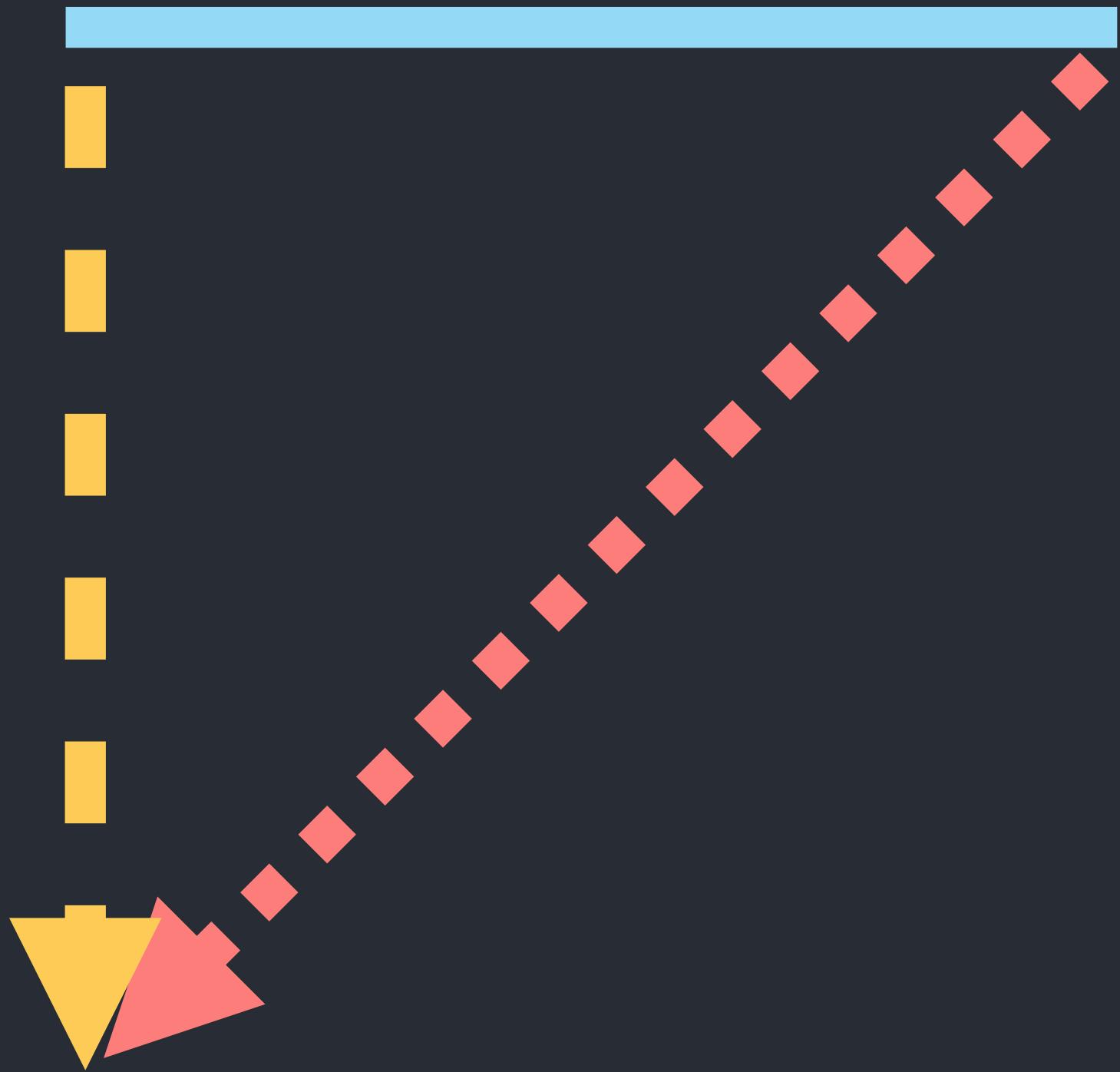




Constructor prototype

new

instance



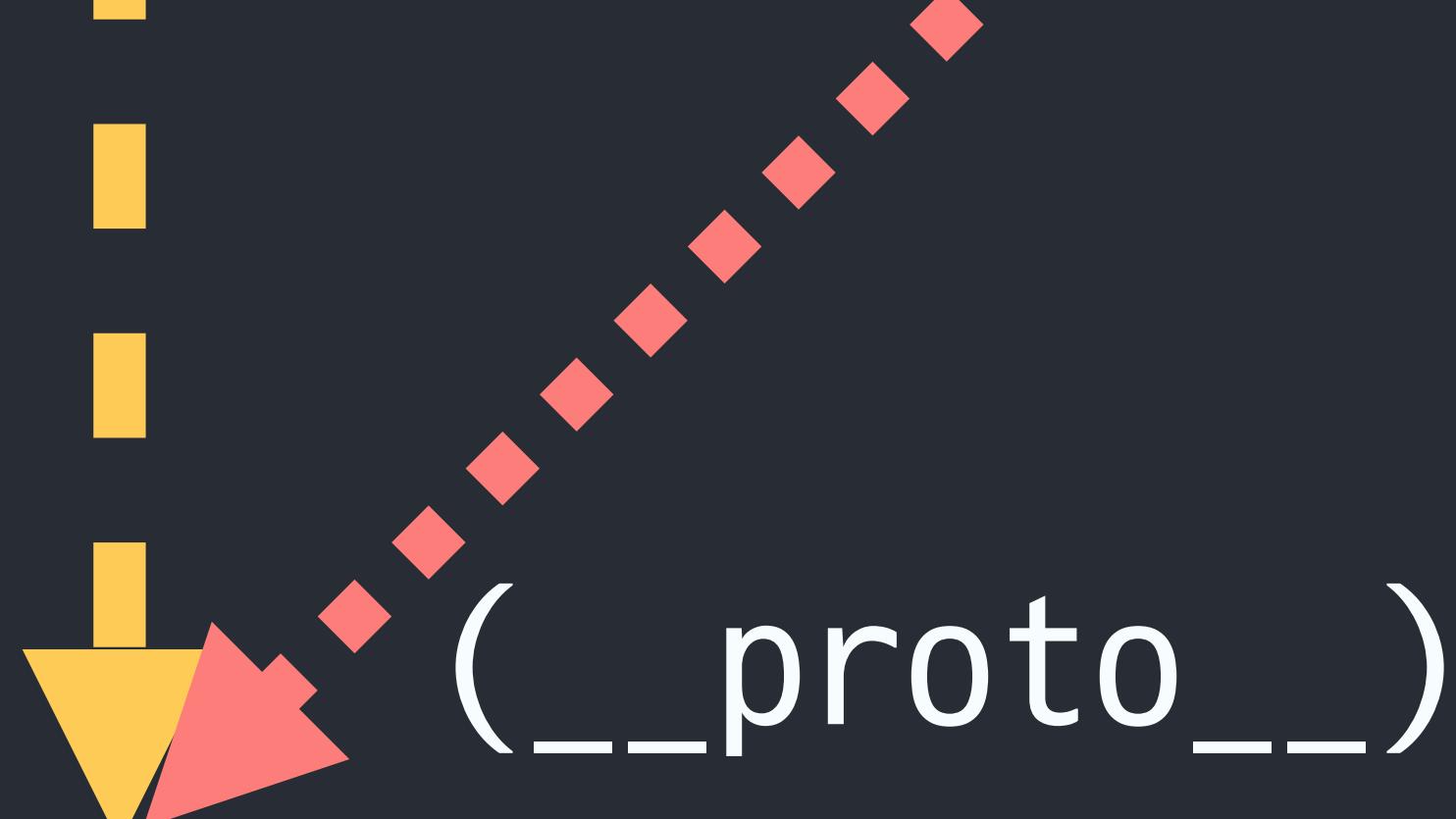
Array

new

from()  
isArray()  
of()  
arguments  
length  
name

prototype

concat()  
filter()  
forEach()  
map()  
push()  
pop()



```
> console.dir([1, 2, 3]);
```

```
▼ Array(3) ⓘ
```

```
 0: 1
```

```
 1: 2
```

```
 2: 3
```

```
 length: 3
```

```
 ► __proto__: Array(0)
```

```
> [1, 2, 3].constructor
```

```
< f Array() { [native code] }
```

```
> [1, 2, 3].__proto__.constructor
```

```
< f Array() { [native code] }
```

```
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()
```

```
// 5-1-1.  
function Person(n, a) {  
    this.name = n;  
    this.age = a;  
}  
  
var gomu = new Person('고무곰', 30);  
  
var gomuClone1 = new gomu.__proto__.constructor('고무곰_클론1', 10);  
  
var gomuClone2 = new gomu.constructor('고무곰_클론2', 25);  
  
var gomuProto = Object.getPrototypeOf(gomu);  
var gomuClone3 = new gomuProto.constructor('고무곰_클론3', 20);  
  
var gomuClone4 = new Person.prototype.constructor('고무곰_클론4', 15);
```

[instance].\_\_proto\_\_  
[instance]  
Object.getPrototypeOf([instance])  
[CONSTRUCTOR].prototype

```
// 5-1-1.  
function Person(n, a) {  
    this.name = n;  
    this.age = a;  
}  
  
var gomu = new Person('고무곰', 30);  
  
[CONSTRUCTOR]  
[instance].__proto__.constructor  
[instance].constructor  
(Object.getPrototypeOf([instance])).constructor  
[CONSTRUCTOR].prototype.constructor  
  
var gomuClone1 = new gomu.__proto__.constructor('고무곰_클론1', 10);  
  
var gomuClone2 = new gomu.constructor('고무곰_클론2', 25);  
  
var gomuProto = Object.getPrototypeOf(gomu);  
var gomuClone3 = new gomuProto.constructor('고무곰_클론3', 20);  
  
var gomuClone4 = new Person.prototype.constructor('고무곰_클론4', 15);
```

5-2.

메소드 상속 및 동작 원리

```
// 5-2-1.  
function Person(n, a) {  
    this.name = n;  
    this.age = a;  
}  
  
var gomu = new Person('고무곰', 30);  
var iu = new Person('아이유', 25);  
  
gomu.setOlder = function() {  
    this.age += 1;  
}  
gomu.getAge = function() {  
    return this.age;  
}  
iu.setOlder = function() {  
    this.age += 1;  
}  
iu.getAge = function() {  
    return this.age;  
}
```

```
// 5-2-2.

function Person(n, a) {
  this.name = n;
  this.age = a;
}
Person.prototype.setOlder = function() {
  this.age += 1;
}
Person.prototype.getAge = function() {
  return this.age;
}

var gomu = new Person('고무곰', 30);
var iu = new Person('아이유', 25);

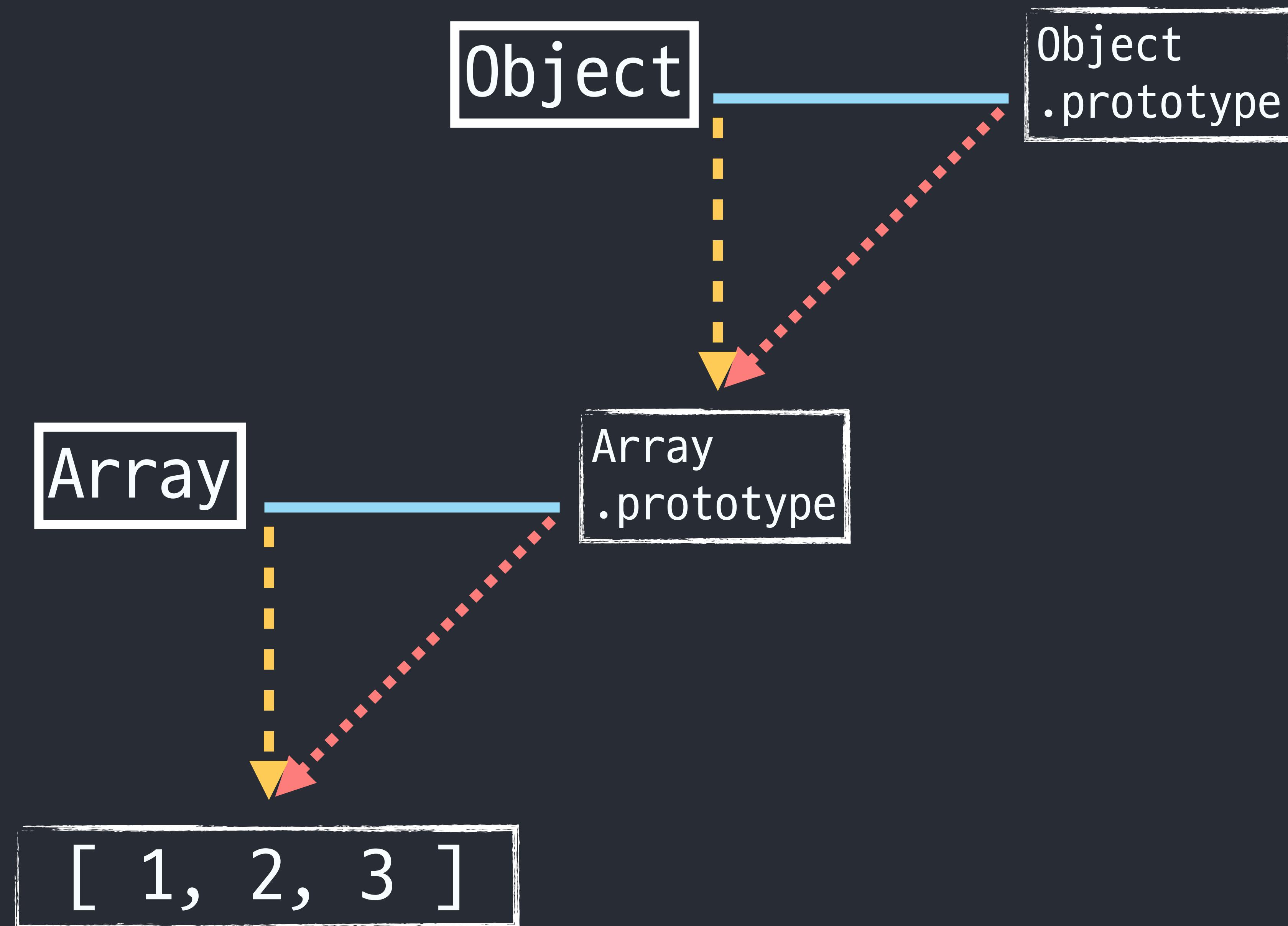
gomu.setOlder();
gomu.getAge(); 31

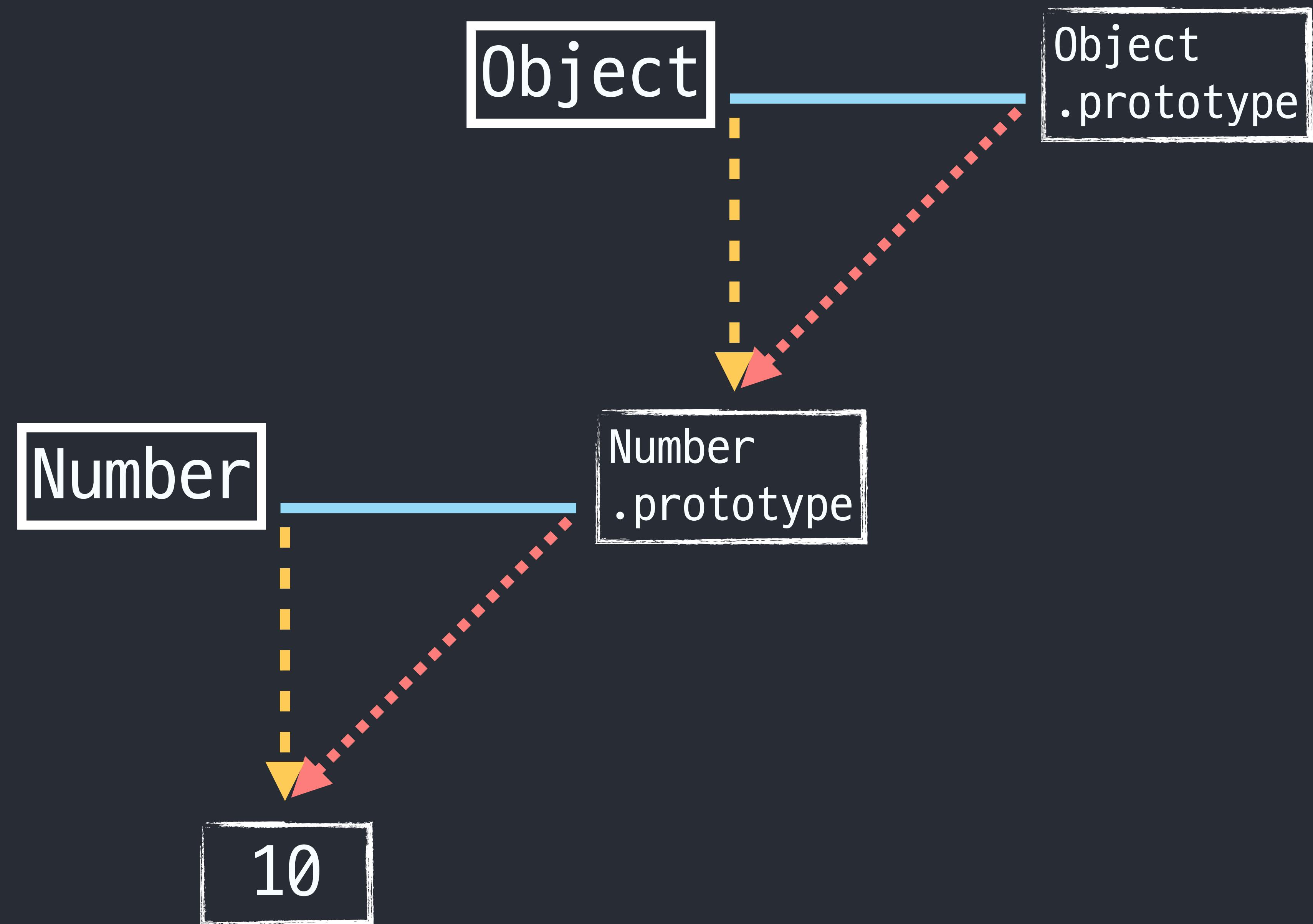
gomu.__proto__.setOlder();
gomu.__proto__.getAge() NaN

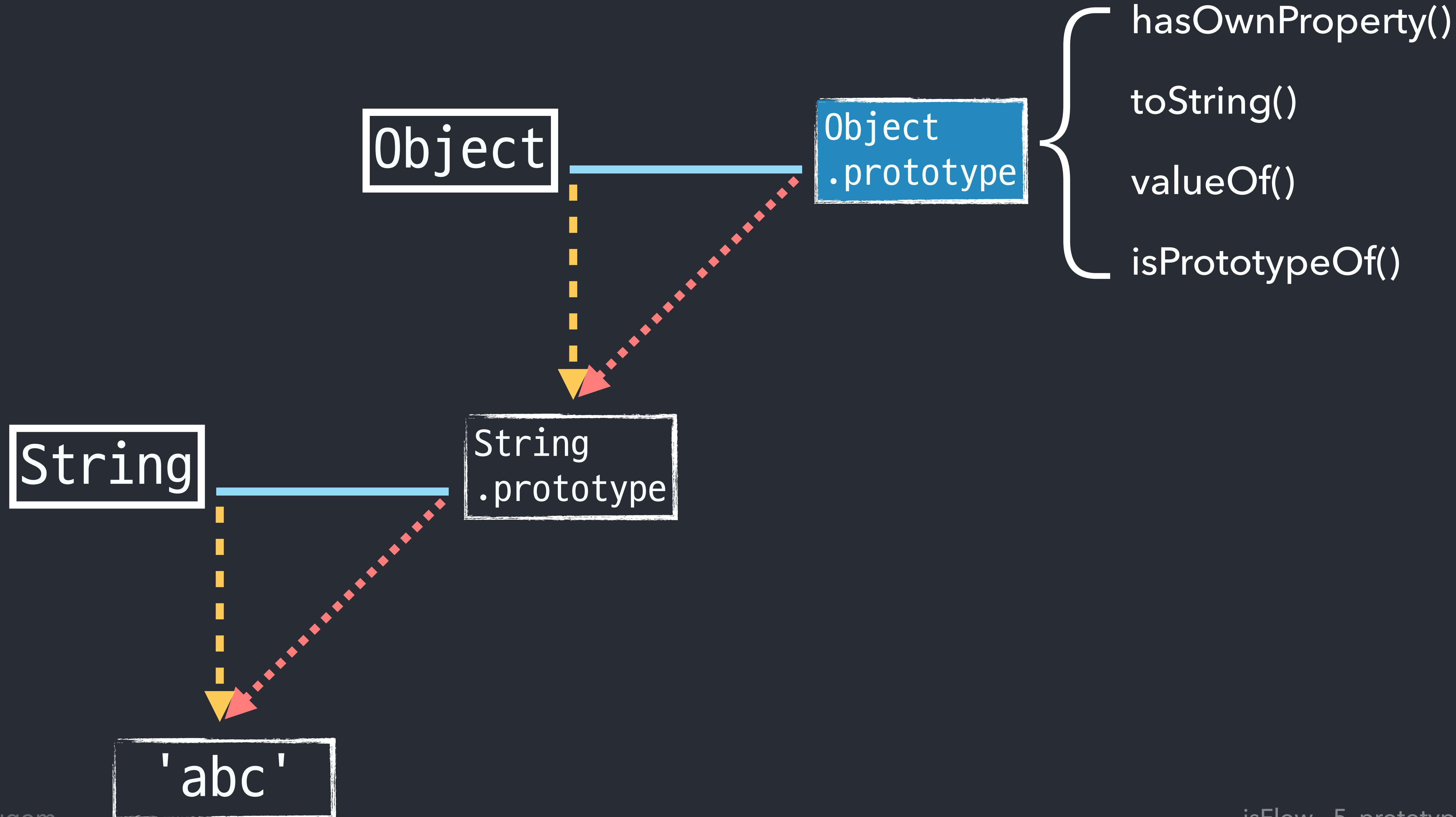
Person.prototype.age = 100;
gomu.__proto__.setOlder();
gomu.__proto__.getAge(); 101
```

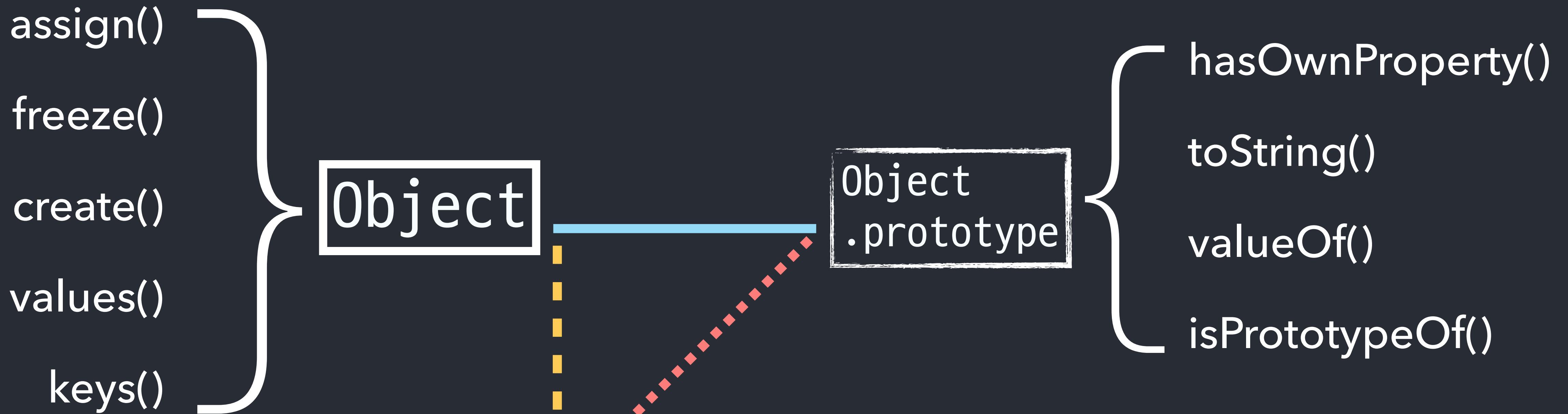
5-3.

# PROTOTYPE CHAINING









{ a: 1, b: 2 }

```
// 5-3-0
```

```
var arr = [1, 2, 3];
```

```
console.log(arr.toString());
```

1,2,3

```
// 5-3-0  
var arr = [1, 2, 3];  
arr.toString = function() {  
    return this.join('_');  
}
```

```
console.log(arr.toString());
```

1\_2\_3

```
// 5-3-0  
var arr = [1, 2, 3];  
arr.toString = function() {  
    return this.join('_');  
}
```

```
console.log(arr.toString());
```

1\_2\_3

```
console.log(arr.__proto__.toString.call(arr));
```

1,2,3

```
console.log(arr.__proto__.__proto__.toString.call(arr));
```

[object Array]

```
// 5-3-0  
var arr = [1, 2, 3];  
Array.prototype.toString = function() {  
    return '[' + this.join(', ') + ']';  
}
```

```
console.log(arr.toString());
```

[1, 2, 3]

```
console.log(arr.__proto__.toString.call(arr));
```

[1, 2, 3]

```
console.log(arr.__proto__.__proto__.toString.call(arr));
```

[object Array]

```
// 5-3-1
var obj = {
  a: 1,
  b: {
    c: 'c'
  }
};
console.log(obj.toString()); [object Object]
```

```
// 5-3-1
var obj = {
  a: 1,
  b: {
    c: 'c'
  },
  toString: function() {
    var res = [];
    for(var key in this) {
      res.push(key + ':' + this[key].toString());
    }
    return '{' + res.join(', ') + '}';
  }
};
console.log(obj.toString());
```

```
{a: 1, b: [object Object], toString: function () {
  var res = [];
  for(var key in this) {
    res.push(key + ':' + this[key].toString());
  }
  return '{' + res.join(', ') + '}';
}}
```

```
// 5-3-1
var obj = {
  a: 1,
  b: {
    c: 'c'
  }
};

Object.prototype.toString = function() {
  var res = [];
  for(var key in this) {
    res.push(key + ': ' + this[key].toString());
  }
  return '{' + res.join(', ') + '}';
}

console.log(obj.toString());
```

{a: 1, b: {c: c}}

```
// 5-3-2
var obj = {
  a: 1,
  b: {
    c: 'c'
  },
  d: [5, 6, 7],
  e: function(){}
};

Object.prototype.toString = function() {
  var res = [];
  for(var key in this) {
    res.push(key + ': ' + this[key].toString());
  }
  return '{' + res.join(', ') + '}';
}

Array.prototype.toString = function() {
  return '[' + this.join(', ') + ']';
}

console.log(obj.toString());
```

```
{a: 1, b: {c: c}, d: [5, 6, 7], e: function (){}}
```

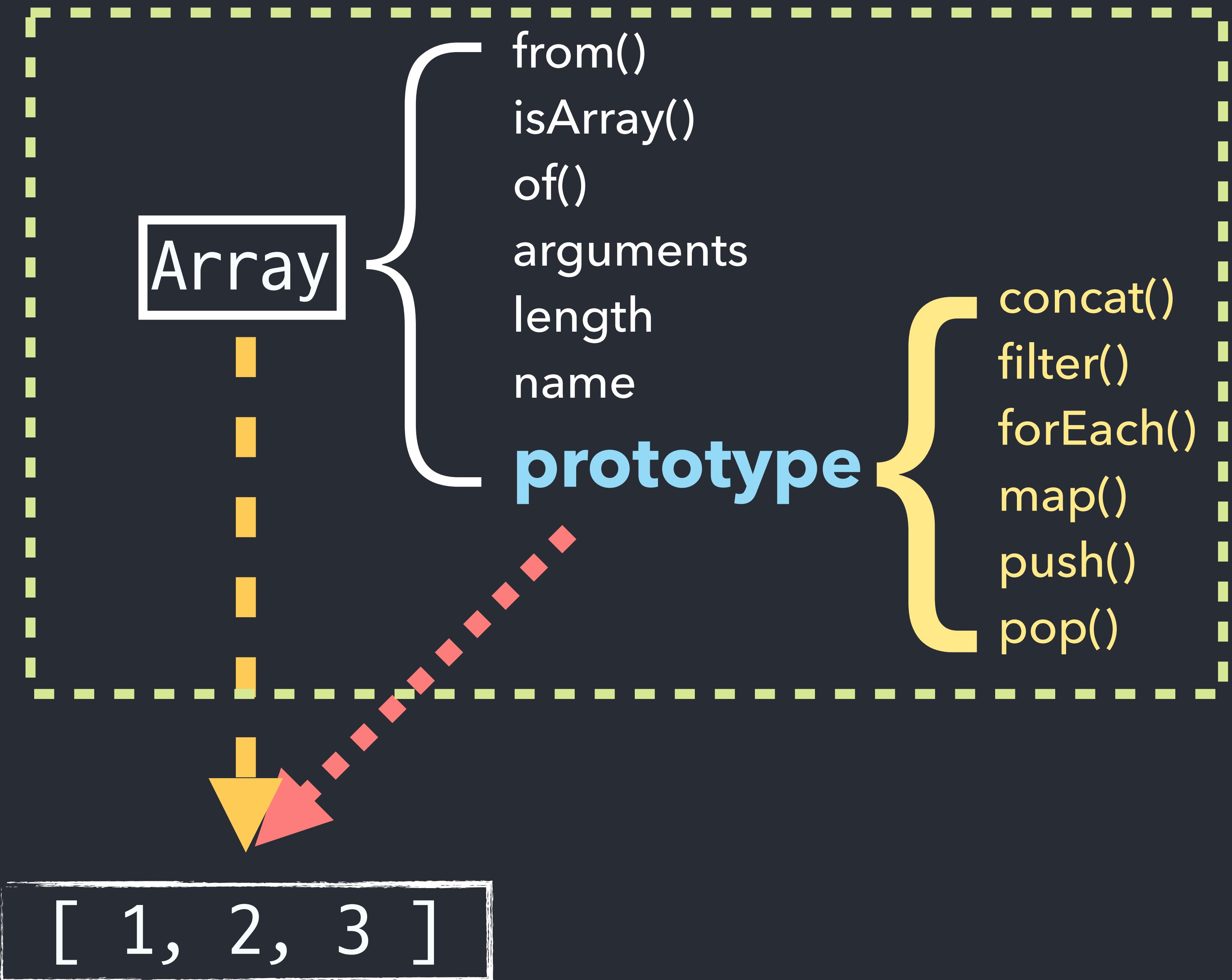
jsFlow 2017. 9. 9. gomugom

---

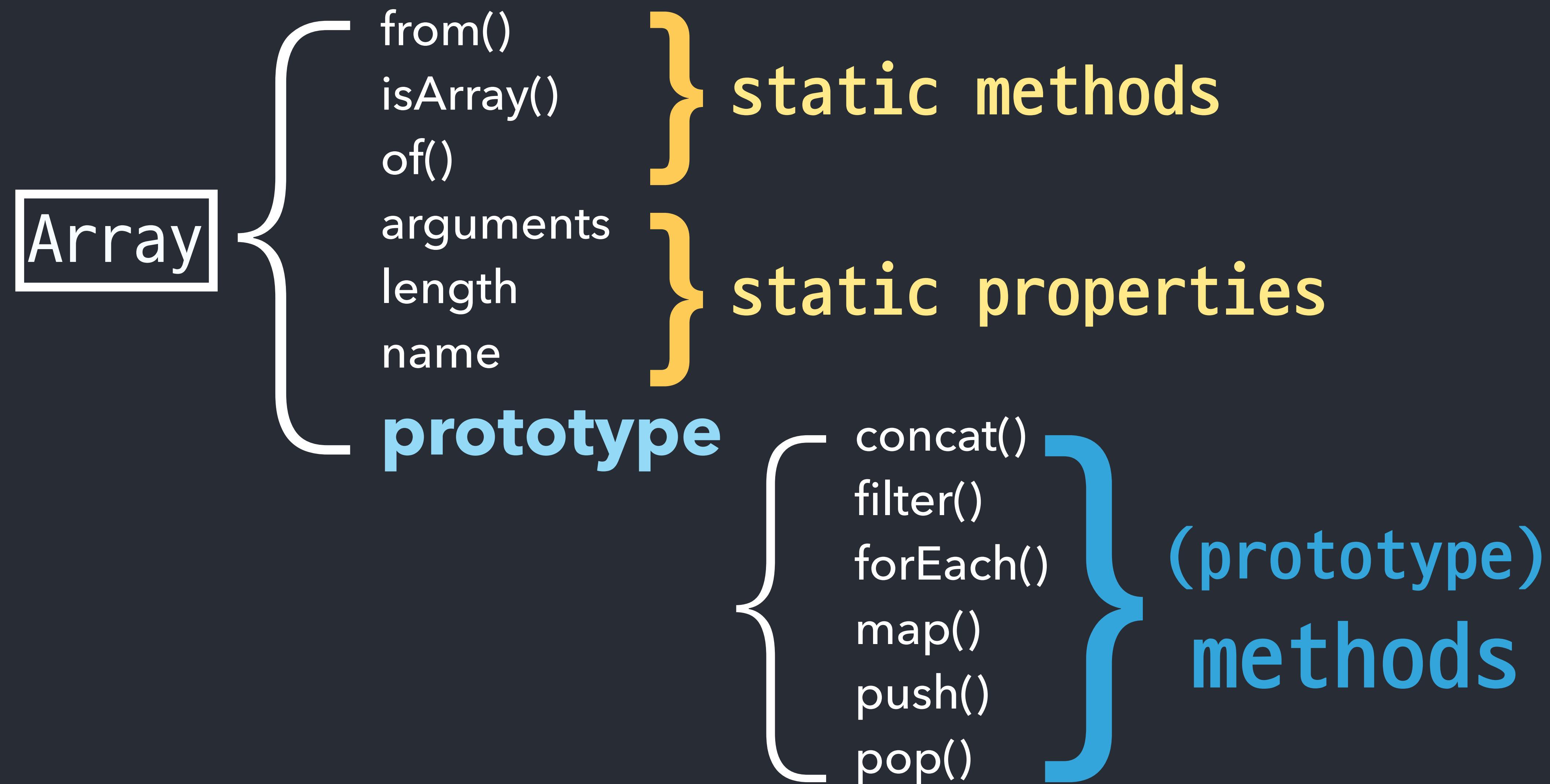
# 6. CLASS

# 6-1. CLASS

(계급. 집단. 집합)



# Class

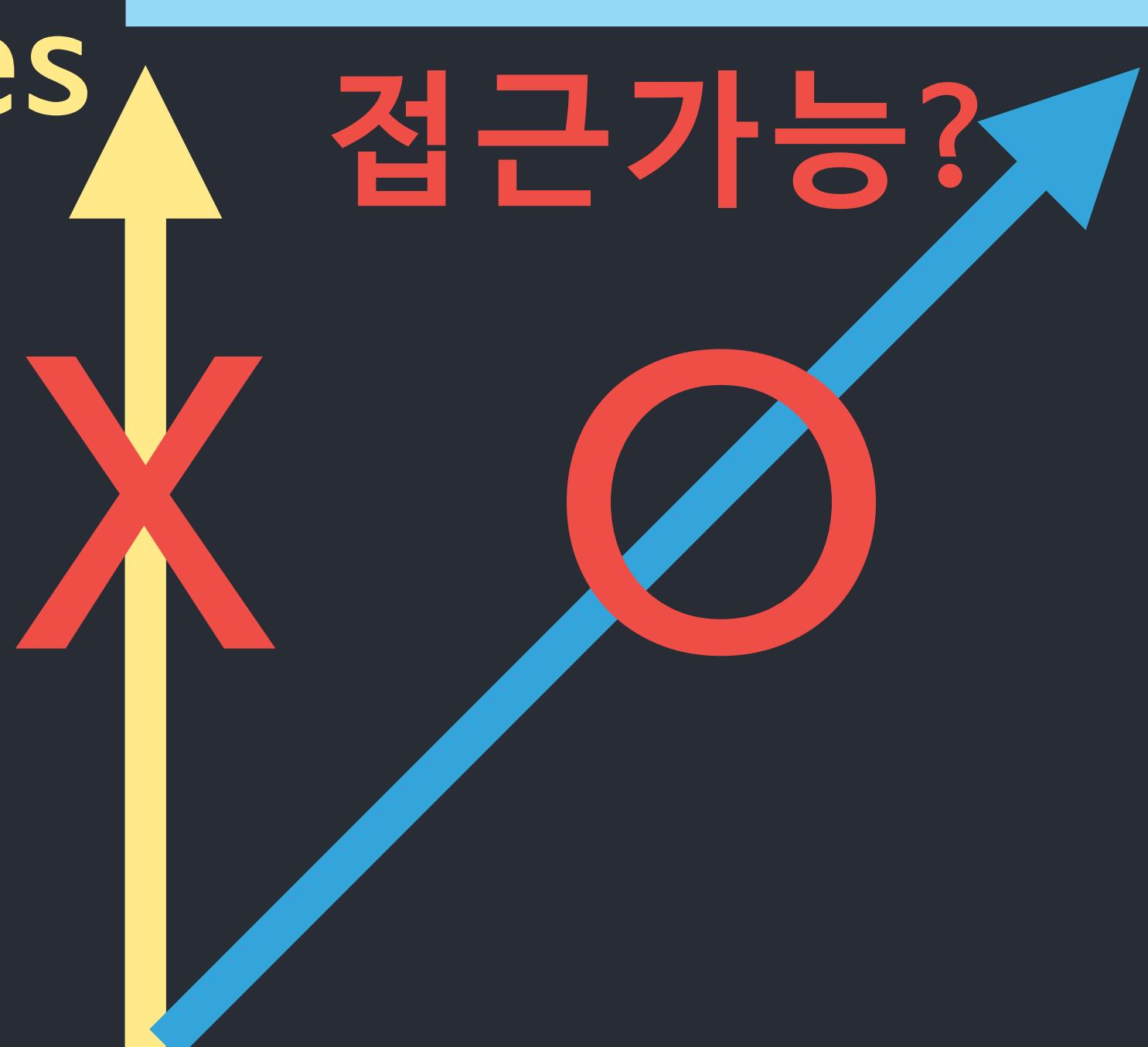


static methods  
static properties

Class

methods  
properties

접근 가능?



instance

**static method**

**(prototype) method**

**(prototype) method**

```
function Person(name, age) {  
    this._name = name;  
    this._age = age;  
}  
  
Person.getInformations = function(instance) {  
    return {  
        name: instance._name,  
        age: instance._age  
    };  
}  
  
Person.prototype.getName = function() {  
    return this._name;  
}  
  
Person.prototype.getAge = function() {  
    return this._age;  
}
```

**static method**

**method**

**method**

```
function Person(name, age) { ... }  
Person.getInformations = function(instance) { ... }  
Person.prototype.getName = function() { ... }  
Person.prototype.getAge = function() { ... }
```

```
var gomu = new Person('고무', 30);
```

**OK**

```
console.log(gomu.getName());
```

**OK**

```
console.log(gomu.getAge());
```

**ERROR**

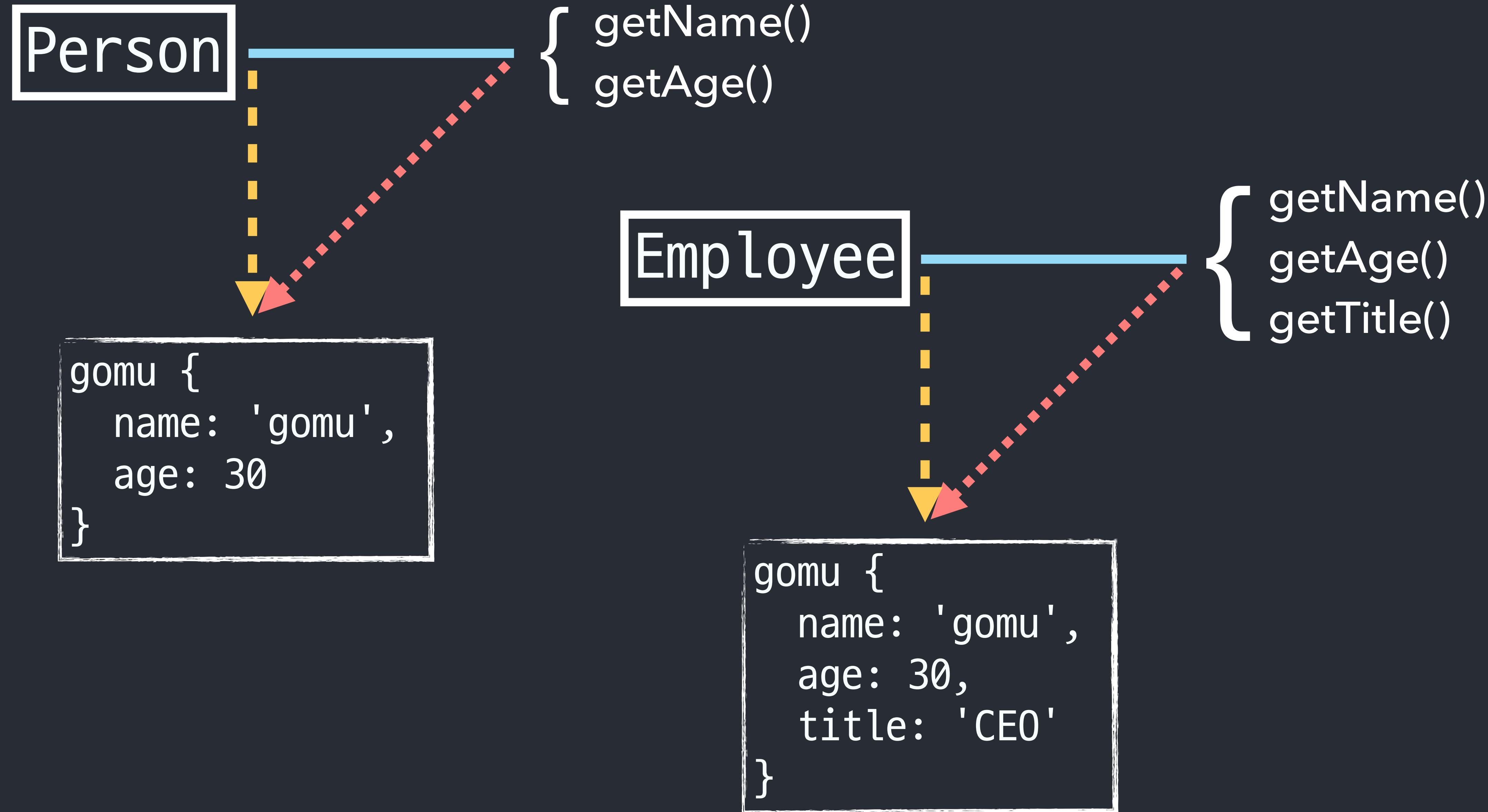
```
console.log(gomu.getInformations(gomu));
```

**OK**

```
console.log(Person.getInformations(gomu));
```

6-2.

# CLASS INHERITANCE



```
// 6-2-1
```

```
function Person(name, age) {  
    this._name = name || '이름없음';  
    this._age = age || '나이모름';  
}
```

```
Person.prototype.getName = function() {  
    return this._name;  
}
```

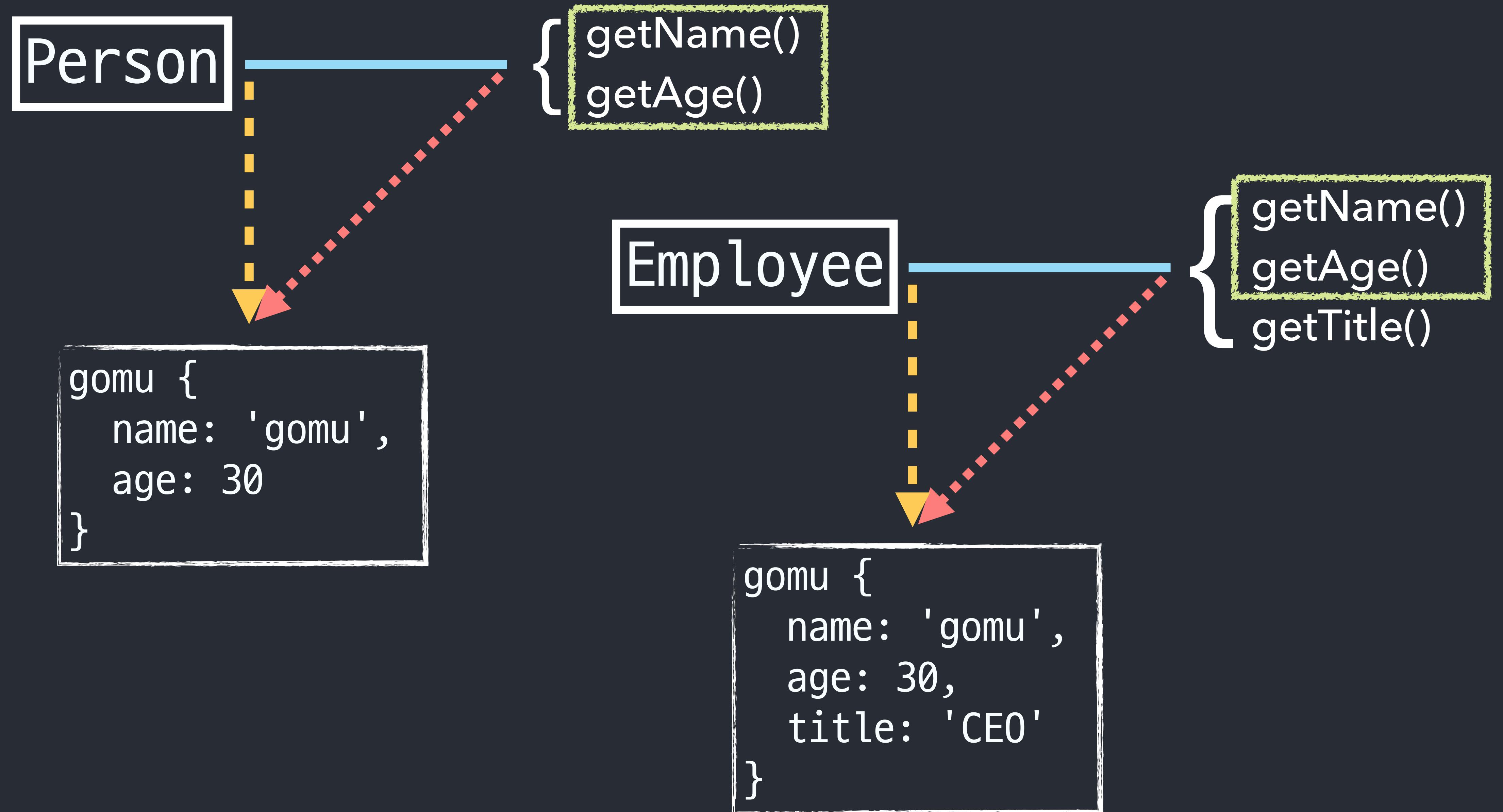
```
Person.prototype.getAge = function() {  
    return this._age;  
}
```

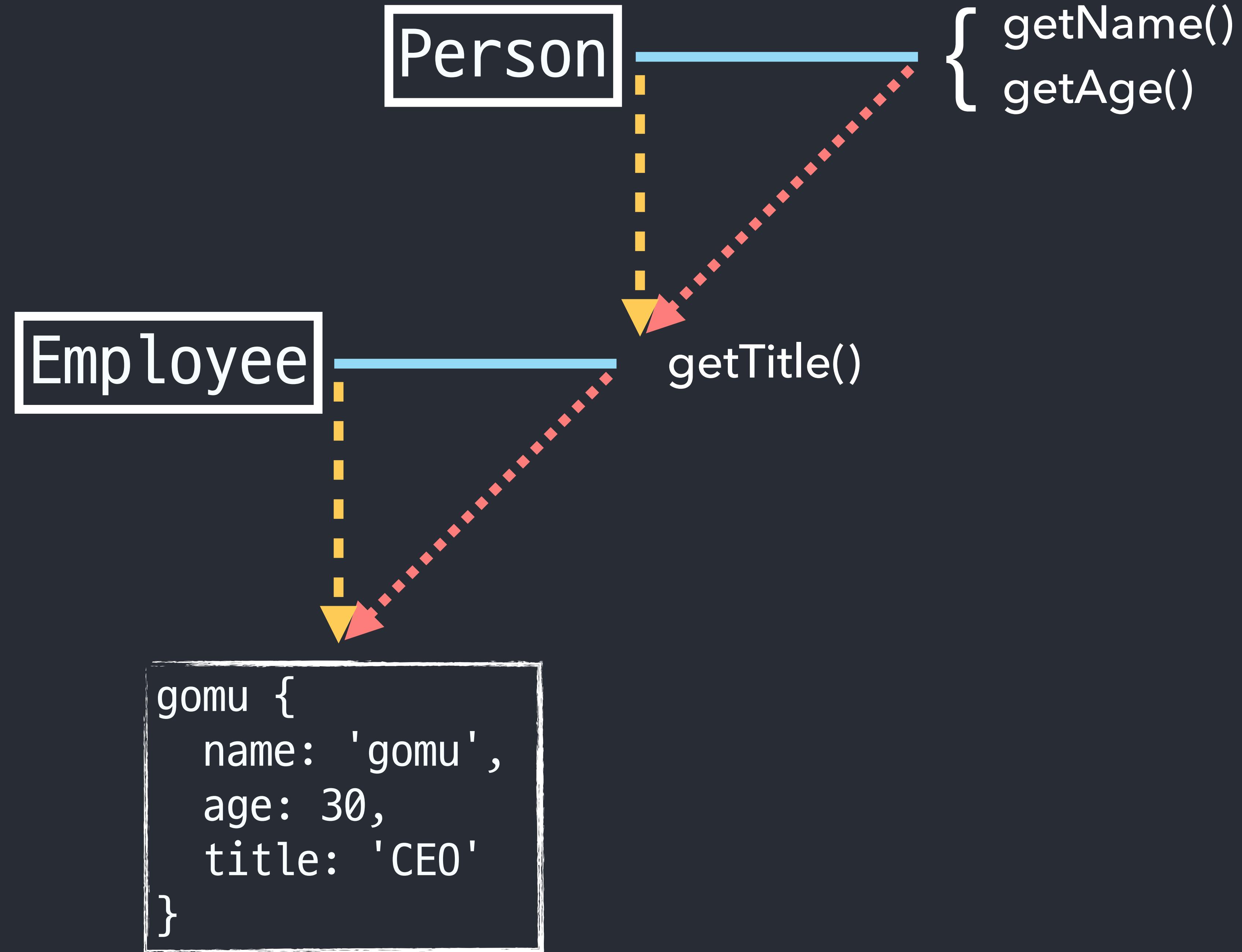
```
function Employee(name, age, position) {  
    this._name = name || '이름없음';  
    this._age = age || '나이모름';  
    this._pos = position || '직책모름';  
}
```

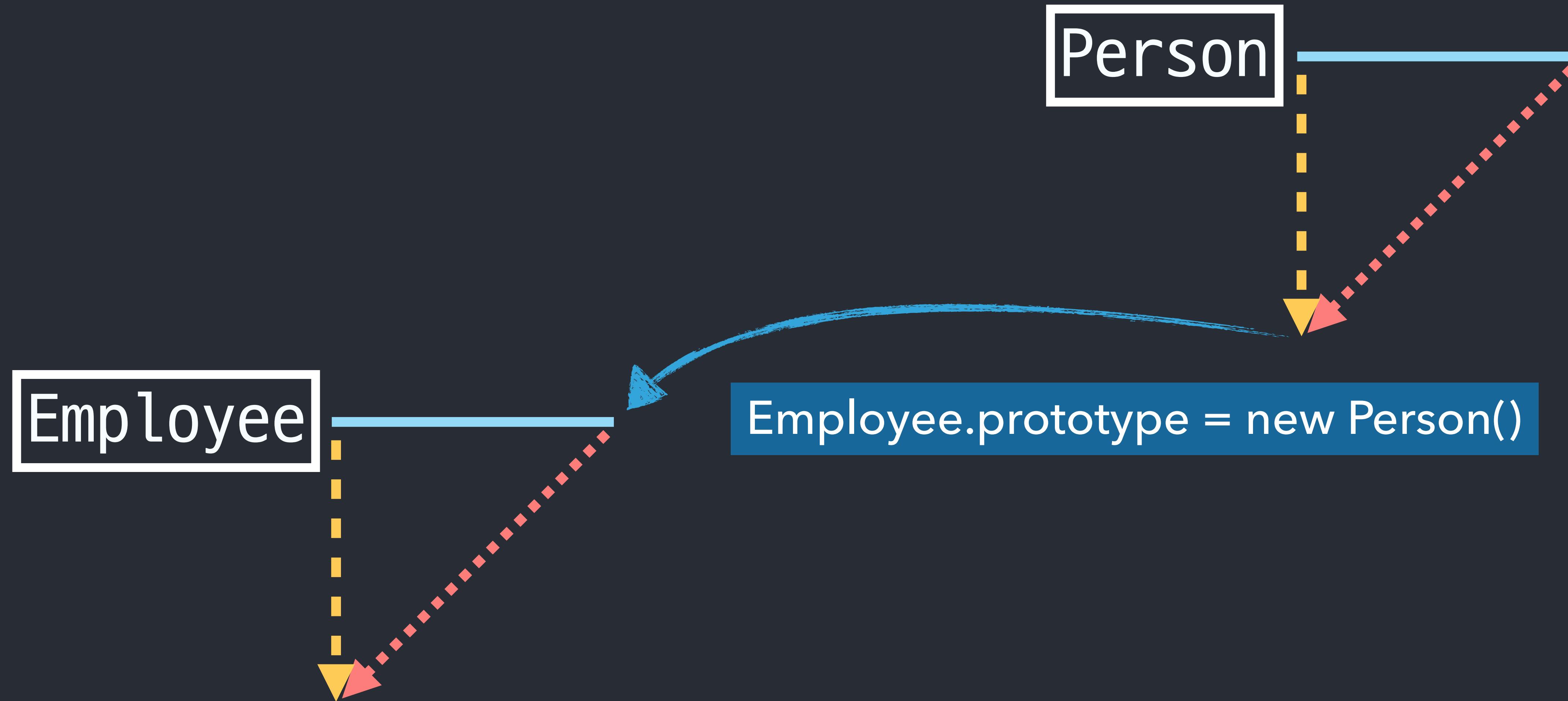
```
Employee.prototype.getName = function() {  
    return this._name;  
}
```

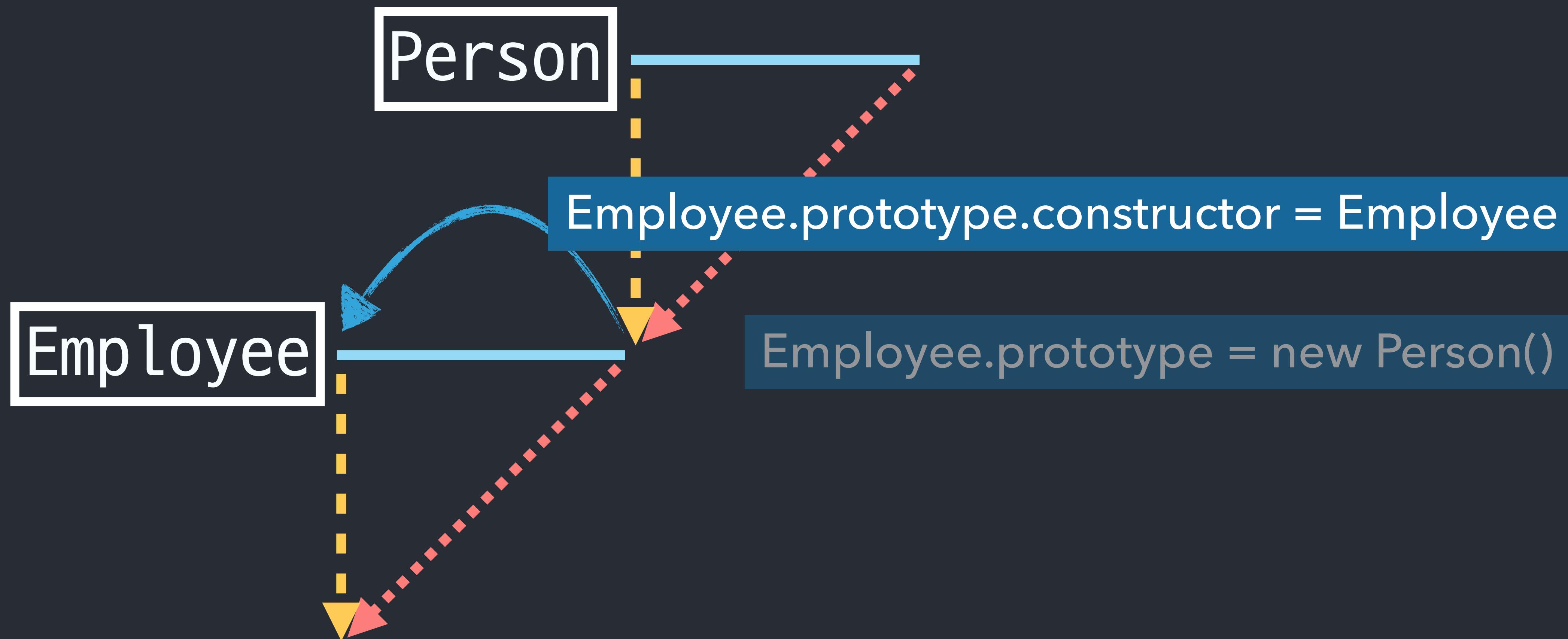
```
Employee.prototype.getAge = function() {  
    return this._age;  
}
```

```
Employee.prototype.getPosition = function() {  
    return this._pos;  
}
```









```
// 6-2-2
function Person(name, age) {
  this._name = name || '이름없음';
  this._age = age || '나이모름';
}
Person.prototype.getName = function() {
  return this._name;
}
Person.prototype.getAge = function() {
  return this._age;
}

function Employee(name, age, position) {
  this._name = name || '이름없음';
  this._age = age || '나이모름';
  this._pos = position || '직책모름';
}

Employee.prototype = new Person();
Employee.prototype.constructor = Employee;
Employee.prototype.getPosition = function() {
  return this._pos;
}
var gomu = new Employee('고무', 30, 'CEO');
console.dir(gomu);

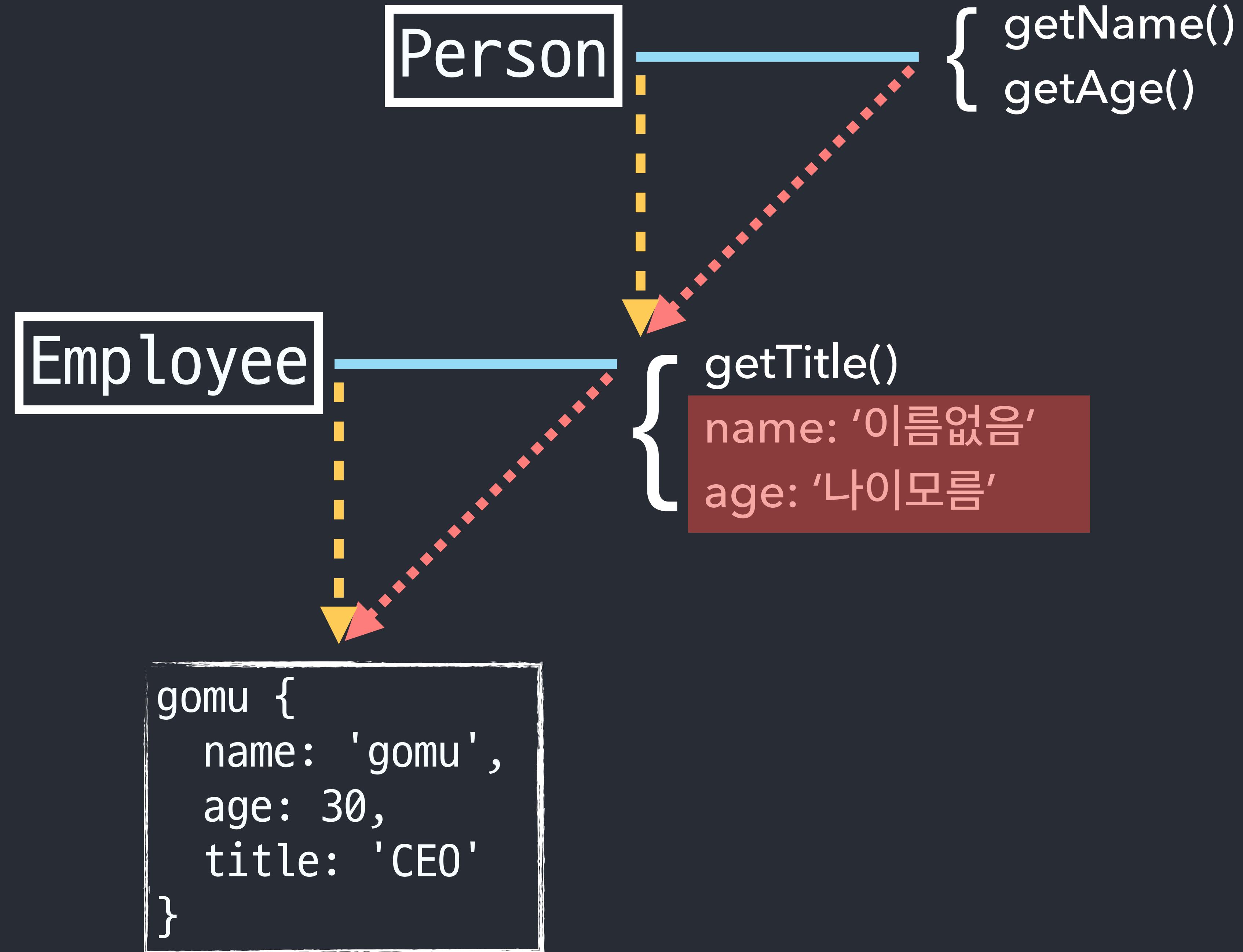
▼ Employee ⓘ
  _age: 30
  _name: "고무"
  _pos: "CEO"
▼ __proto__: Person
  ► constructor: function Employee(name, age)
  ► getPosition: function ()
    _age: "나이모름"
    _name: "이름없음"
▼ __proto__: Object
  ► getAge: function ()
  ► getName: function ()
  ► constructor: function Person(name, age,
```

gomu

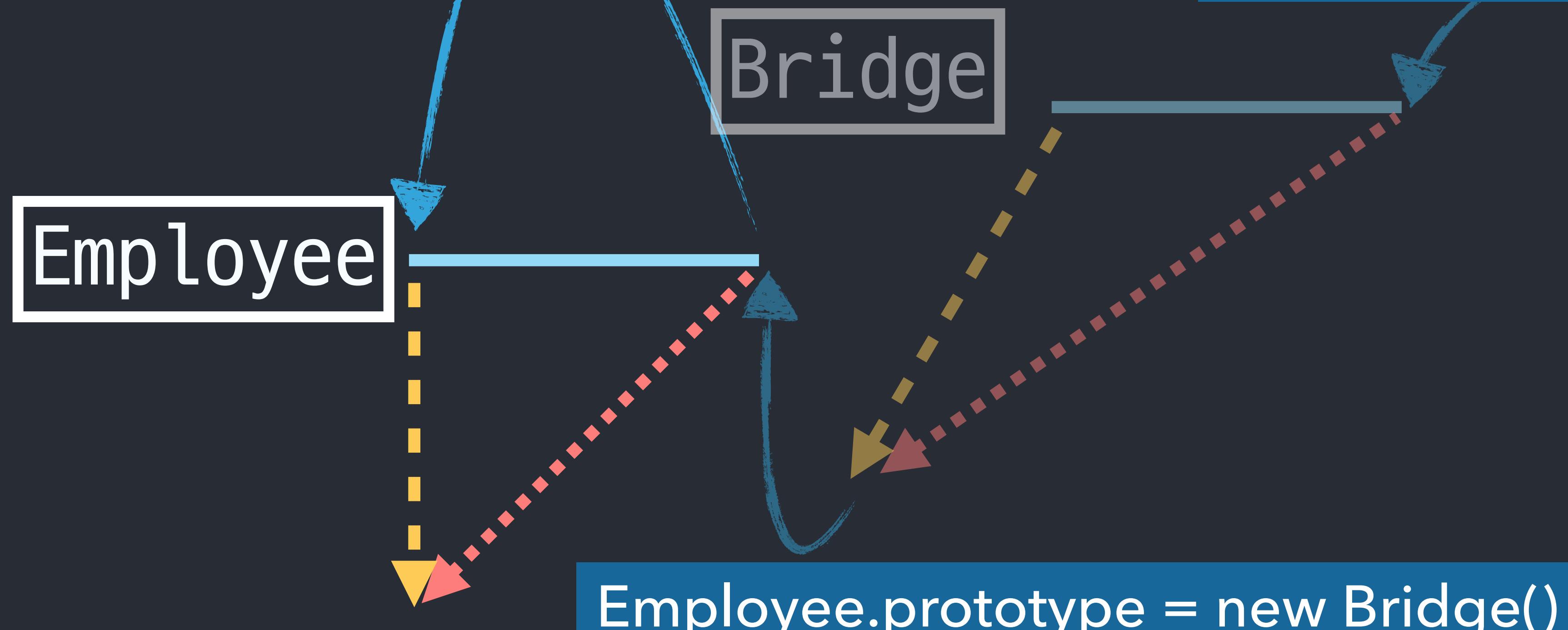
gomu.\_\_proto\_\_  
Employee.prototype

gomu.\_\_proto\_\_.\_\_proto\_\_  
Person.prototype

▼ Employee ⓘ  
  \_age: 30  
  \_name: "고무"  
  \_pos: "CEO"  
▼ \_\_proto\_\_: Person  
  ► constructor: *function Employee(name, age)*  
  ► getPosition: *function ()*  
    \_age: "나이모름"  
    \_name: "이름없음"  
▼ \_\_proto\_\_: Object  
  ► getAge: *function ()*  
  ► getName: *function ()*  
  ► constructor: *function Person(name, age)*  
  ► \_\_proto\_\_: Object



`Employee.prototype.constructor = Employee`



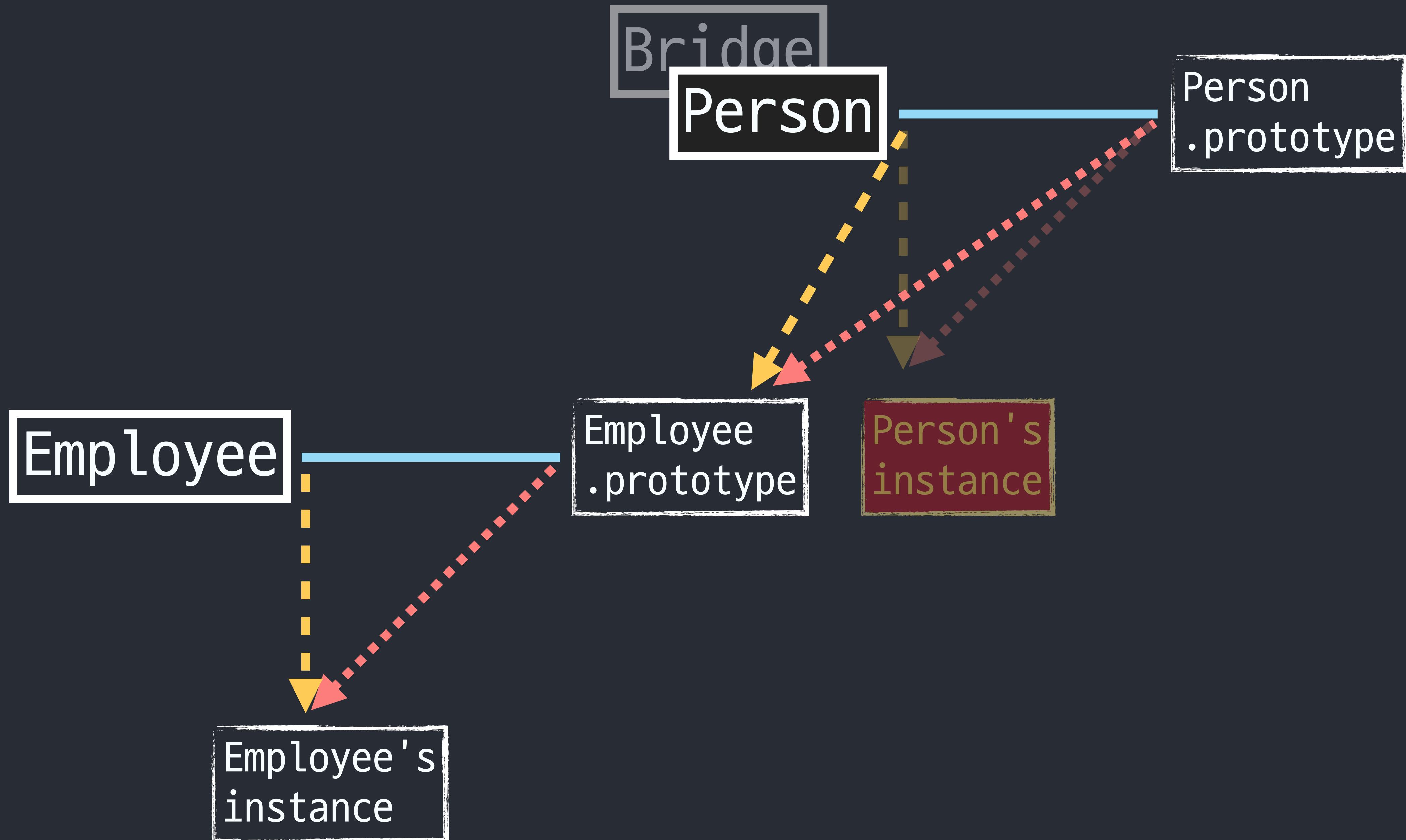
```
// 6-2-2
function Person(name, age) {
  this._name = name || '이름없음';
  this._age = age || '나이모름';
}
Person.prototype.getName = function() {
  return this._name;
}

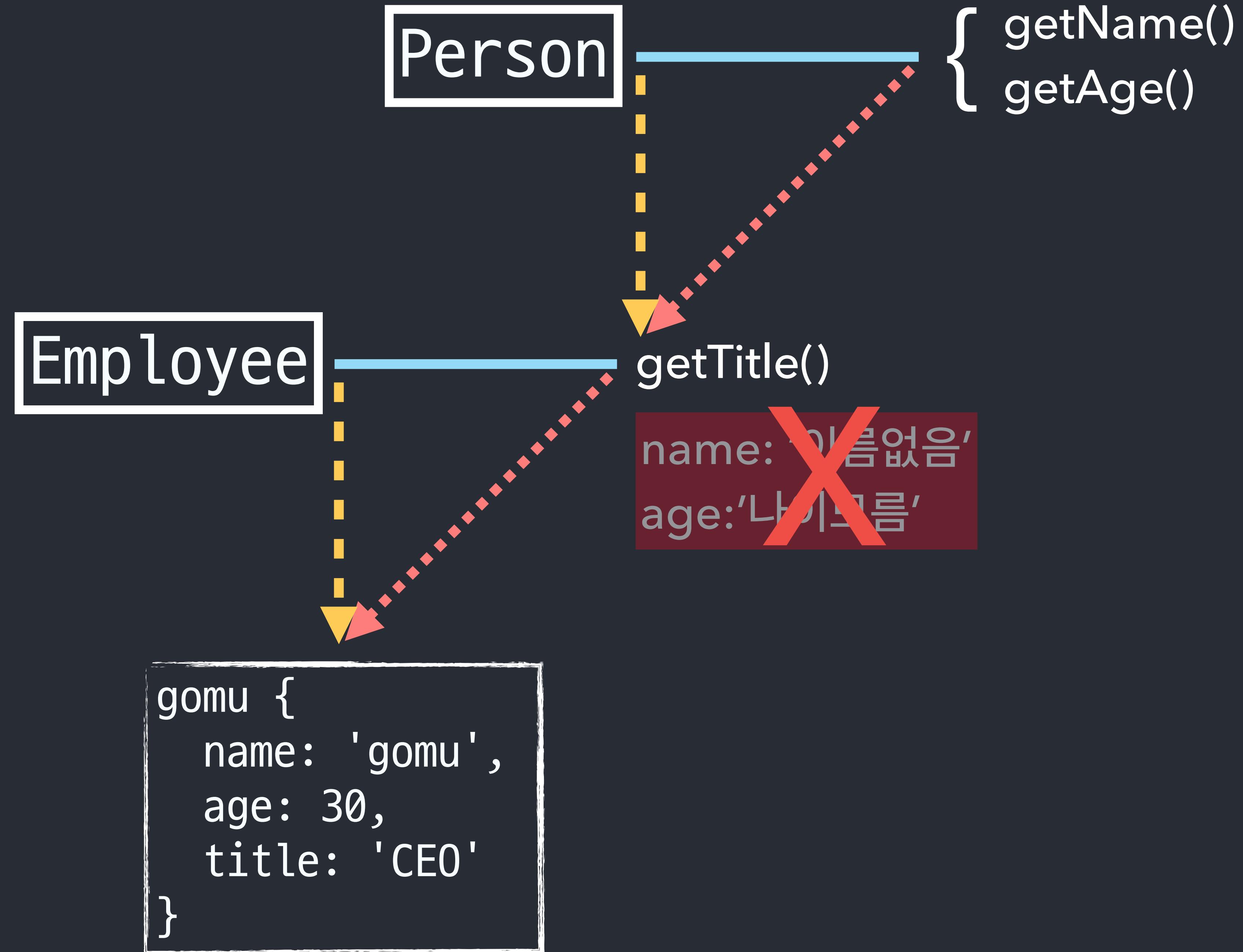
Employee i
  _age: 30
  _name: "고무"
  _pos: "CEO"
  ▶ __proto__: Person
    ► constructor: function Employee()
    ► getPosition: function ()
  ▶ __proto__: Object
    ► getAge: function ()
    ► getName: function ()
    ► constructor: function Person()
    ► __proto__: Object
```

```
function Bridge() {}
Bridge.prototype = Person.prototype;
Employee.prototype = new Bridge();
Employee.prototype.constructor = Employee;

Employee.prototype.getPosition = function(){
  return this._pos;
}

var gomu = new Employee('고무', 30, 'CEO');
console.dir(gomu);
```





```
// 6-2-2
function Person(name, age) {
    this._name = name || '이름없음';
    this._age = age || '나이모름';
}
Person.prototype.getName = function() {
    return this._name;
}
Person.prototype.getAge = function() {
    return this._age;
}

function Employee(name, age, position) {
    this._name = name || '이름없음';
    this._age = age || '나이모름';
    this._pos = position || '직책모름';
}

var extendClass = (function() {
    function Bridge(){}
    return function(Parent, Child) {
        Bridge.prototype = Parent.prototype;
        Child.prototype = new Bridge();
        Child.prototype.constructor = Child;
        Child.superClass = Parent.prototype;
    }
})();

extendClass(Person, Employee);
Employee.prototype.getPosition = function(){
    return this._pos;
}

var gomu = new Employee('고무', 30, 'CEO');
console.dir(gomu);
```