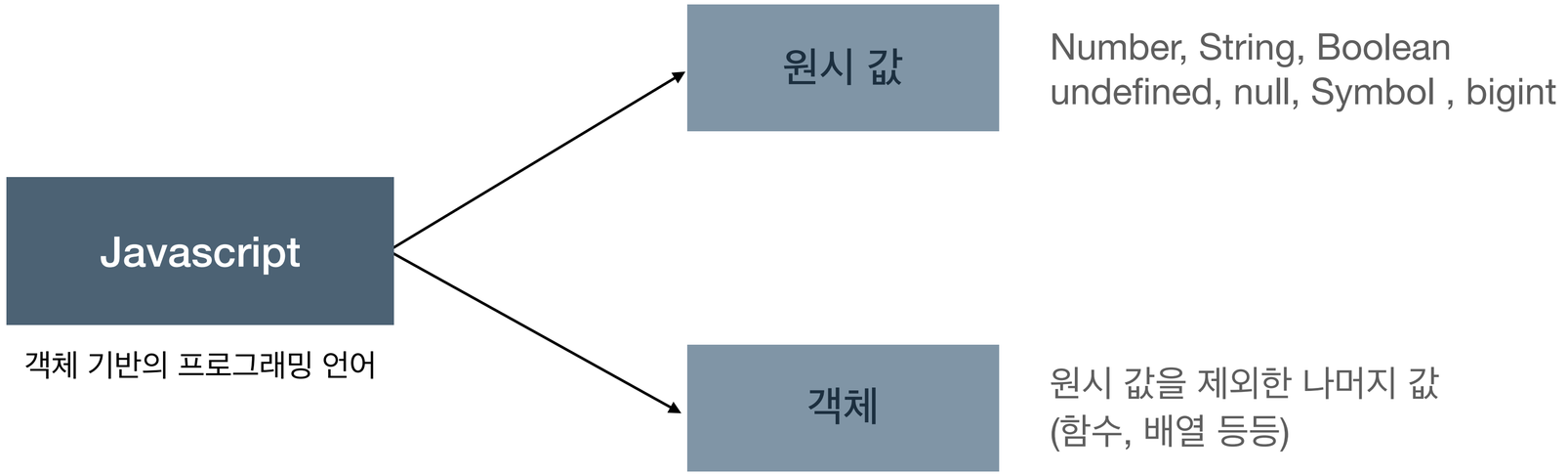
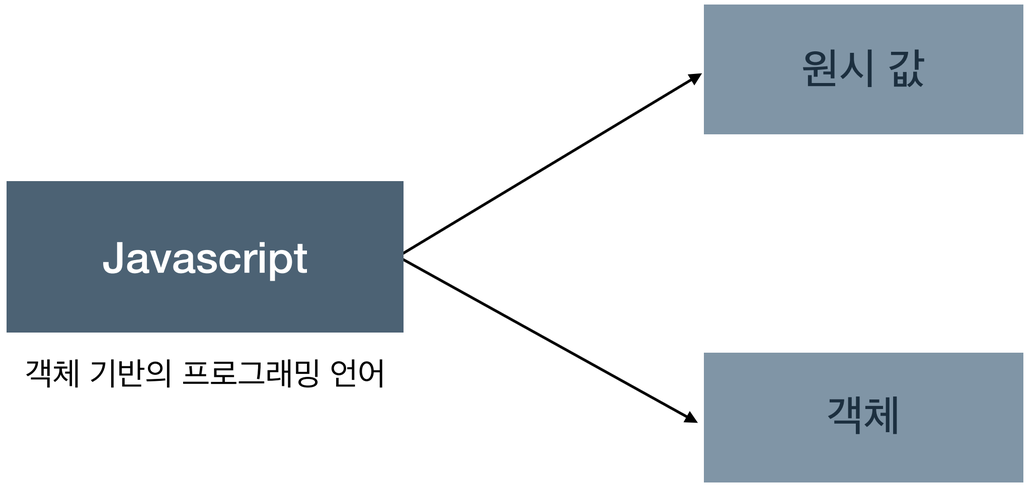
생성자 함수

20230413 박재영

객체란


Javascript
객체 기반의 프로그래밍 언어
원시 값
Number, String, Boolean
undefined, null, Symbol , bigint
객체
원시 값을 제외한 나머지 값
(함수, 배열 등등)

객체란



객체 기반의 프로그래밍 언어

하나의 값
변경 불가능한 값

다양한 타입의 값을 하나의 단위로 구성한 복합적인 자료
변경 가능한 값

객체란

- 객체
  - 프로퍼티의 집합
- 프로퍼티
  - 키와 값으로 구성

```
const person = {
  프로퍼티 키 : 프로퍼티 값,
  name : 'Park',
  age    :  27,
}
```

프로퍼티

객체란

- 메서드
  - 프로퍼티 값이 함수인 프로퍼티

```
const person = {
  name : 'Park',
  age    :  27,
  sayHello : function ( ) {
     console.log('hello world');
  }
}
```

프로퍼티

메서드

객체란

- 객체
  - 프로퍼티의 집합

```
const person = {
  name : 'Park',
  age    :  27,
  sayHello : function ( ) {
    console.log('hello world');
  }
}
```

프로퍼티 — 객체의 상태를 나타내는 데이터 (특성)

메서드 — 객체가 가지고 있는 어떤 동작

객체 리터럴

객체 리터럴에 의한 객체 생성

Javascript → 프로토타입 기반 객체 지향 언어

객체 생성 방법

- 객체 리터럴
- 생성자 함수
- Object.create 메서드
- 클래스

객체 리터럴에 의한 객체 생성

Javascript → 프로토타입 기반 객체 지향 언어

객체 생성 방법

- 객체 리터럴
- 생성자 함수
- Object.create 메서드
- 클래스

객체 리터럴에 의한 객체 생성

리터럴 표기법

- 사람이 이해할 수 있는 문자나 기호로 값을 생성하는 표기법

리터럴

- 데이터 그 자체

```
const person = {
    name : 'Park',
    age    :  27,
    sayHello : function ( ) {
        console.log('hello world');
    }
}
```

객체 리터럴에 의한 객체 생성

객체 리터럴로 객체를 만든다

- 객체 데이터 그 자체를 정해진 문법대로 직접 기술하여 객체를 생성한다

```
const person = {
    name : 'Park',
    age    :  27,
    sayHello : function ( ) {
        console.log('hello world');
    }
}
```

객체 리터럴에 의한 객체 생성

```
const person = {
  name : 'Park',
  age    :  27,
  sayHello : function ( ) {
     console.log('hello world');
  }
}
```

프로퍼티

프로퍼티

- **객체는 프로퍼티의 집합**
- 키와 값으로 구성되어 있으며 프로퍼티를 나열할 때 쉼표로 구분한다
- 프로퍼티 키는 값에 접근할 수 있는 식별자 역할을 한다
- 식별자 네이밍 규칙을 준수하지 않아도 괜찮지만
  차이가 존재하기 때문에 식별자 네이밍 규칙을 준수할 것을 권장한다

객체 리터럴에 의한 객체 생성

```
const person = {
    name : 'Park',
    age    :  27,
    sayHello : function ( ) {
        console.log('hello world');
    }
}
```

메서드

메서드

- **객체는 프로퍼티의 집합**
- 함수는 값으로 취급될 수 있다
- 프로퍼티 값으로 함수가 사용되면 그 프로퍼티를 메서드라고 부른다

프로퍼티 조작

프로퍼티 조작

프로퍼티 접근

```
const person = {
    name : 'Park',
    age    :  27,
    sayHello : function ( ) {
        console.log('hello world');
    }
}
```

마침표 표기법

```
console.log( person.name )
```

→ Park

대괄호 표기법

```
console.log( person['age'] )
```

→ 27

프로퍼티 조작

프로퍼티 접근

```
const person = {
    name : 'Park',
    age    :  27,
    sayHello : function ( ) {
        console.log('hello world');
    }
}
```

대괄호 표기법에 문자열 형식으로 넣지 않은 경우

```
console.log( person[name] )
```

→ ReferenceError

객체에 없는 프로퍼티에 접근한 경우

```
console.log( person.gender )
```

→ undefined

프로퍼티 조작

프로퍼티 값 갱신

```
person.name = 'Lee'
person.age = '스물일곱'
```

```
person = {
    name : 'Lee',
    age    : '스물일곱',
    sayHello : function ( ) {
        console.log('hello world');
    }
}
```

프로퍼티 조작

프로퍼티 동적 생성

```
person.likeJS = false
person['likeFriday'] = false
```

```
person = {
    name : 'Lee',
    age    : '스물일곱',
    sayHello : function ( ) {
        console.log('hello world');
    },
    likeJS :  false,
    likeFriday : false
}
```

프로퍼티 조작

프로퍼티 삭제

```
delete person.likeJS
delete person['likeFriday']
```

```
person = {
    name : 'Lee',
    age    : '스물일곱',
    sayHello : function ( ) {
        console.log('hello world');
    }
}
```

생성자 함수 (constructor)

생성자 함수 (constructor)

생성자 함수

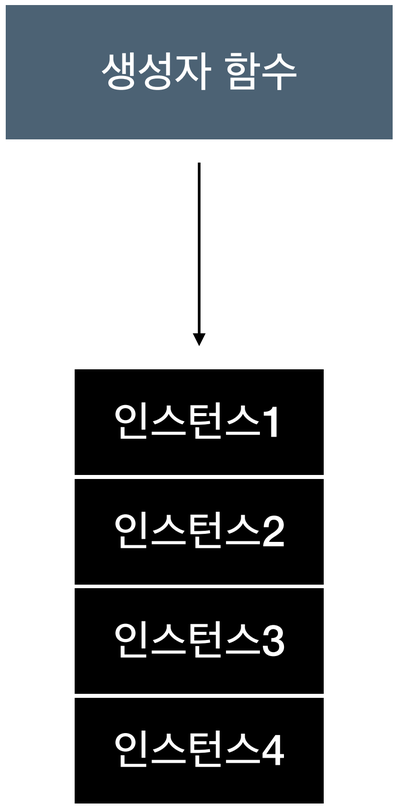
- new 연산자와 함께 호출해서 객체를 생성하는 함수
- new 연산자와 함께 생성자 함수를 호출하면 객체를 생성해서 반환한다

인스턴스

- 생성자 함수에 의해 생성된 객체

new 연산자

- new 연산자는 내장 객체 타입 혹은 사용자 정의 객체타입의 인스턴스를 생성한다



생성자 함수 (constructor)

빌트인 생성자 함수

Object 생성자 함수

- Object( ) -> new 연산자와 함께 호출하면 그냥 빈 객체를 만든다

자바스크립트는 Object 생성자 함수 외에도 여러 빌트인 생성자 함수가 제공된다

- String( )
- Number( )
- Boolean( )
- Array( )
- Map ( )
- 등등..

생성자 함수 (constructor)

빌트인 생성자 함수

- 예제 person을 Object 생성자 함수로 만들기

```
const person = new Object ( {
   name : 'Park',
   age    :  27,
   sayHello : function ( ) {
      console.log('hello world');
   }
} )
```

```
const person = new Object (  )
// 빈 객체 생성


person.name = 'Park'
person.age = 27
person.sayHello =  function ( ) {
      console.log('hello world');
   }
```

생성자 함수 (constructor)

빌트인 생성자 함수

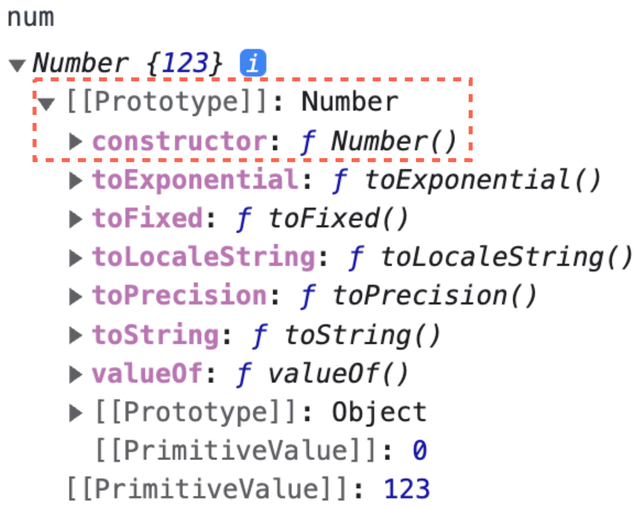
- Number, String, Boolean?

```
var num = Number(123);
var num = Number('123');
typeof num
```

→ 123
Number

```
var num = new Number(123);
var num = new Number('123');
typeof num
```

→ Number {123}
Object



생성자 함수 (constructor)

빌트인 생성자 함수

- Number, String, Boolean?

```
var num = Number(123);
var num = Number('123');
typeof num
```

→ 123 Number

```
var num = new Number(123);
var num = new Number('123');
typeof num
```

→ Number {123} Object

num

Number {123}

[[Prototype]]: Number

toExponential: f toExponential()

toLocaleString: f toLocaleString()

toPrecision: f toPrecision()

valueOf: f valueOf()

[[Prototype]]: Object

[[PrimitiveValue]]: 0

[[PrimitiveValue]]: 123

Number, String, Boolean 함수가 왜 생성자 함수로 존재하는지는
원시값이 객체 타입이 아닌데도 자연스럽게 메서드를 사용할 수 있는것과 연관이 있는 개념

ex) (10).toString()

래퍼 개체(wrapper object)에 대해 공부해보면 도움이 될 것임

생성자 함수 (constructor)

객체 리터럴에 의한 객체 생성 방식의 문제점

```
const person1 = {
    name : 'Park',
    age    :  27,
    sayHello : function ( ) {
        console.log('hello world');
    }
}
```

생성자 함수 (constructor)

객체 리터럴에 의한 객체 생성 방식의 문제점

```
const person1 = {
    name : 'Park',
    age    :  27,
    sayHello : function ( ) {
        console.log('hello world');
    }
}
```

```
const person2 = {
    name : 'Lee',
    age    :  10,
    sayHello : function ( ) {
        console.log('hello world');
    }
}
```

...

```
const person1000 = {
    name : 'Cho',
    age    :  50,
    sayHello : function ( ) {
        console.log('hello world');
    }
}
```

생성자 함수 (constructor)

객체 리터럴에 의한 객체 생성 방식의 문제점

```
const person1 = {
  name : 'Park',
  age  : 27,
  sayHello : function ( ) {
    console.log('hello world');
  }
}
```

```
const person2 = {
  name : 'Lee',
  age  : 10,
  sayHello : function ( ) {
    console.log('hello world');
  }
}
```

...

```
const person1000 = {
  name : 'Cho',
  age  : 50,
  sayHello : function ( ) {
    console.log('hello world');
  }
}
```

프로퍼티는 객체의 상태를 나타내는 데이터라고 할 수 있다.
즉, 프로퍼티는 값이 객체마다 다를 수는 있다.

생성자 함수 (constructor)

객체 리터럴에 의한 객체 생성 방식의 문제점

```
const person1 = {
    name : 'Park',
    age    :  27,
    sayHello : function ( ) {
        console.log('hello world');
    }
}
```

```
const person2 = {
    name : 'Lee',
    age    :  10,
    sayHello : function ( ) {
        console.log('hello world');
    }
}
```

...

```
const person1000 = {
    name : 'Cho',
    age    :  50,
    sayHello : function ( ) {
        console.log('hello world');
    }
}
```

메서드는 객체마다 내용이 동일한 경우가 일반적이다.

생성자 함수 (constructor)

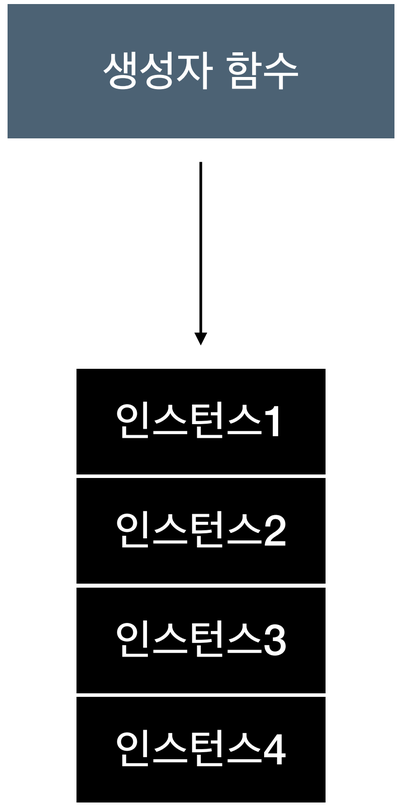
생성자 함수

함수의 장점

- 재사용성
- 유지보수 향상
- 가독성 향상

생성자 함수

- 함수의 장점을 취하면서 객체를 생성할 수 있다.



생성자 함수 (constructor)

생성자 함수

1
```
function Person( name ) {
    this.name = name;
    this.sayHi = function ( ) {
        console.log('Hello! my name is ${this.name}');
    }
}
```

2
```
const person = new Person('Lee');
```

생성자 함수에 의한 객체 생성

1. 함수를 작성하여 객체 타입을 정의
2. new 연산자로 객체의 인스턴스 생성

생성자 함수 (constructor)

생성자 함수

```
function Person( name ) {
    this.name = name;
    this.sayHi = function ( ) {
        console.log('Hello! my name is ${this.name}');
    }
}

const person1 = new Person('Lee');
const person2 = new Person('Park');
// ...
const person1000 = new Person('Cho');
```

생성자 함수에 의한 객체 생성

이와 같은 방식은 객체를 생성할 때 템플릿처럼 생성자 함수를 이용한다

객체 리터럴에 비해 상당히 짧아진 코드로 프로퍼티 구조가

동일한 객체를 여러 개 만들 수 있다

생성자 함수 (constructor)

생성자 함수

```
function Person( name ) {
    this.name = name;
    this.sayHi = function ( ) {
        console.log( `Hello! my name is ${this.name}` );
    }
}

const person = new Person('Lee');
```

this?

this는 객체가 자신의 프로퍼티나 메서드를 참조하기 위한 변수

생성자 함수에서 this는 자신이 앞으로 생성할 인스턴스를 가리킨다

또한, 프로퍼티나 메서드는 this를 통해서
자신이 속한 프로퍼티나 메서드를 참조할 수 있다

생성자 함수 (constructor)

this

메서드는 자신이 속한 객체의 프로퍼티를 참조하고 조작하는 능력이 있어야 한다

이를 수행하려면 자신이 속한 객체를 가리키는 식별자를 참조할 수 있어야 한다.

객체 리터럴의 경우

```
const person1 = {
    name : 'Lee',
    sayHello : function ( ) {
        console.log(`Hello! my name is ${person1.name}`);
        // console.log(`Hello! my name is ${this.name}`);
    }
}
person1.sayHello( ) // Hello! my name is Lee
```

객체 리터럴이 메서드 내부에서 자신이 속한 객체인 person1 식별자를 참조할 수 있는 이유

객체 리터럴이 person1 변수에 할당되기 전에 평가된다

sayHello 메서드가 호출되는 시점에서는 객체 리터럴의 평가가 완료된 상황

다시말해 메서드 호출 시점에는 이미 person1 식별자에 객체가 할당됨

생성자 함수 (constructor)

this

메서드는 자신이 속한 객체의 프로퍼티를 참조하고 조작하는 능력이 있어야 한다

이를 수행하려면 자신이 속한 객체를 가리키는 식별자를 참조할 수 있어야 한다.

객체 리터럴의 경우

```
const person1 = {
    name : 'Lee',
    sayHello : function ( ) {
        console.log(`Hello! my name is ${person1.name}`);
        // console.log(`Hello! my name is ${this.name}`);
    }
}
person1.sayHello( ) // Hello! my name is Lee
```

일반적이지 않고, 좋은 방법은 아니다.

생성자 함수 (constructor)

```
function Person( name ) {
    ????.name = name;
    ????.sayHi = function ( ) {
        console.log('Hello! my name is ${????.name}');
    }
}
// 이 상황에서는 생성자 함수가 자신이 생성할 인스턴스에 대한 정보를 모름

const person = new Person('Lee');
// 생성자 함수로 인스턴스를 만들려면 생성자 함수가 정의되어야 한다
```

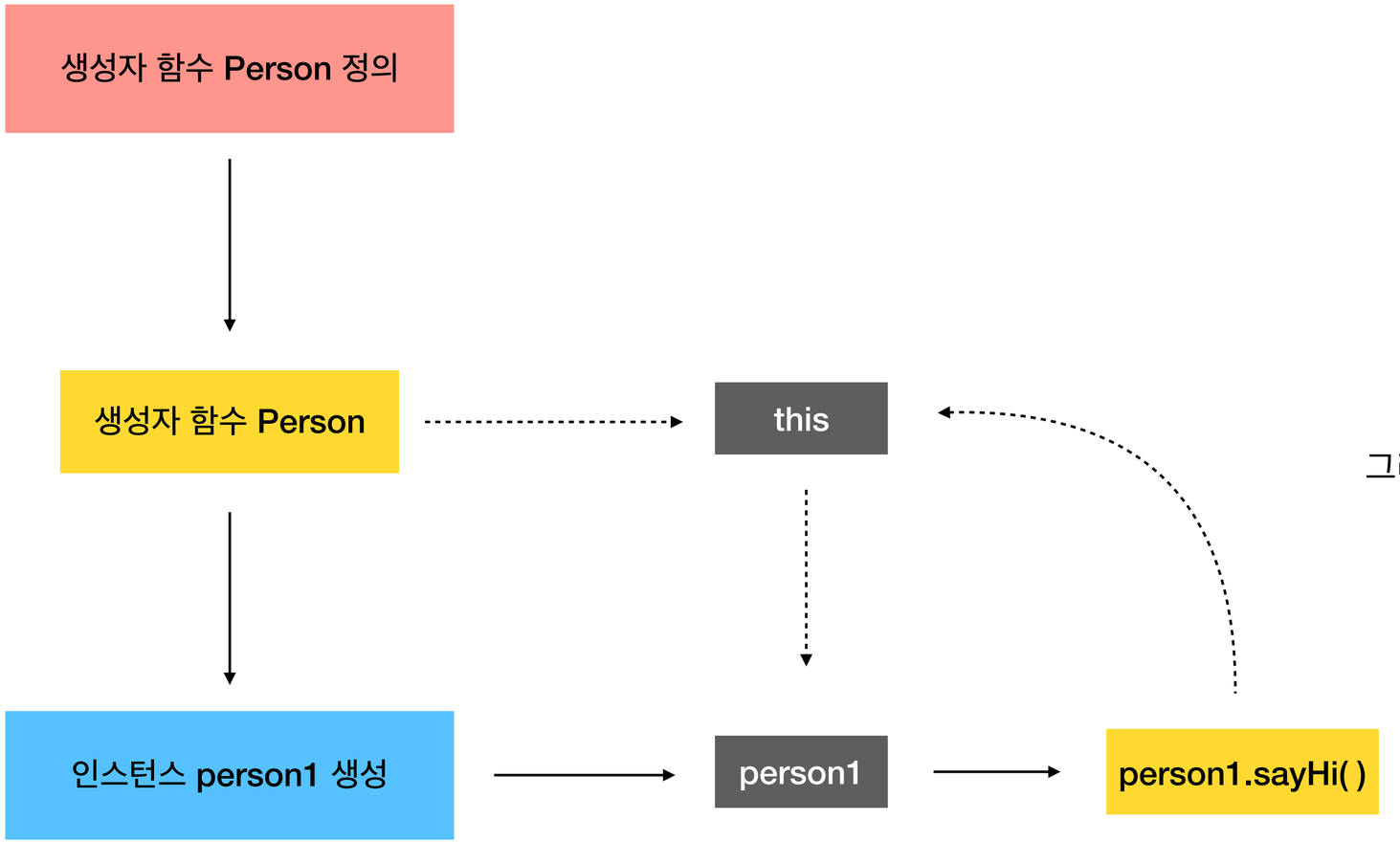
생성자 함수를 정의할 때도 똑같이 하면 되는거 아닌가?

생성자 함수로 인스턴스를 생성하려면, 생성자 함수의 정의와 호출이 필요하다
그러나 이 시점에선 인스턴스가 만들어지기 전이다

객체 리터럴 방식과 다르게 생성자 함수는
자신이 만들 인스턴스의 식별자를 알 수 없다

그렇기에 생성할 인스턴스를 참조할 수 없다.

생성자 함수 (constructor)



그래서 this가 제공된다

생성자 함수에서 this는 자신이 속한 객체나 자신이 생성할 인스턴스를 가리킨다

그리고 this를 통해서 자신이 속한 인스턴스의 프로퍼티와 메서드를 참조할 수 있다

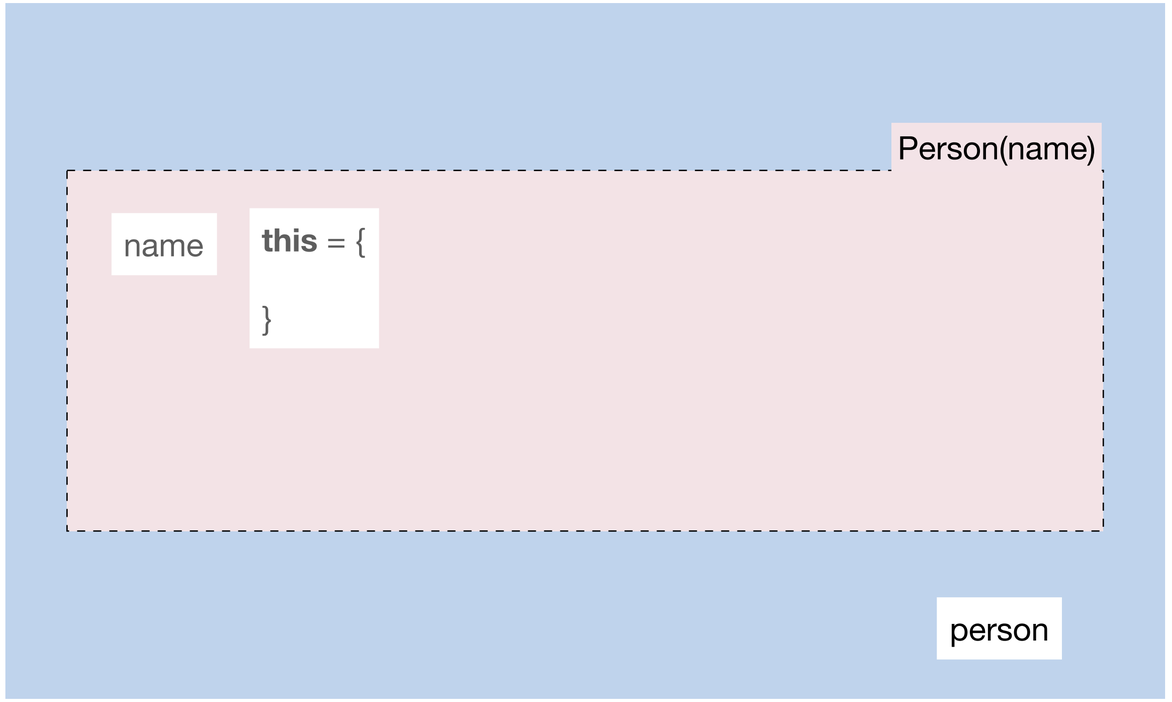
생성자 함수 (constructor)

생성자 함수 인스턴스 생성 과정

```
function Person(name) {
    // 런타임 이전에 암묵적으로 빈 객체가 생성되고 this에 연결된다
    this.name = name;
    this.sayHi = function ( ) {
        console.log('Hello! my name is ${this.name}');
    }
    // new 키워드로 호출된 생성자 함수는 암묵적으로 this를 반환한다
}

const person = new Person('Lee');
```


const person = new Person('Lee');

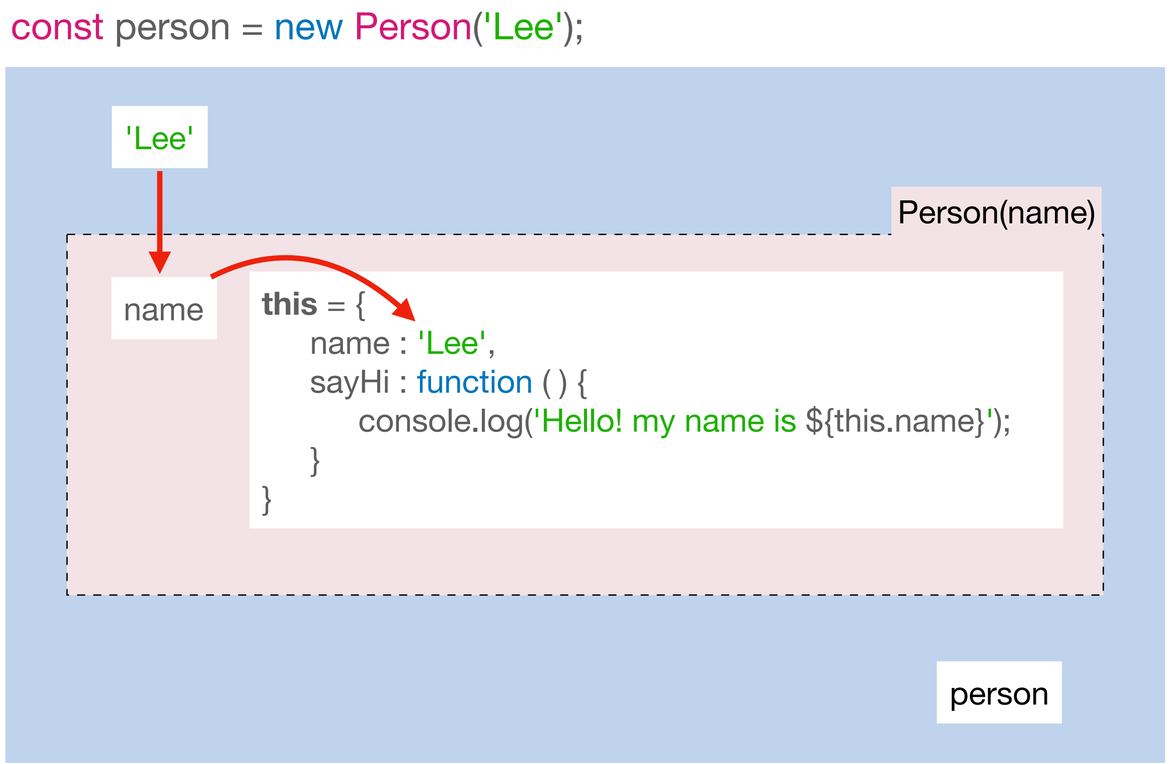


생성자 함수 (constructor)

생성자 함수 인스턴스 생성 과정

```
function Person(name) {
    // 런타임 이전에 암묵적으로 빈 객체가 생성되고 this에 연결된다
    this.name = name;
    this.sayHi = function ( ) {
        console.log('Hello! my name is ${this.name}');
    }
    // new 키워드로 호출된 생성자 함수는 암묵적으로 this를 반환한다
}

const person = new Person('Lee');
```

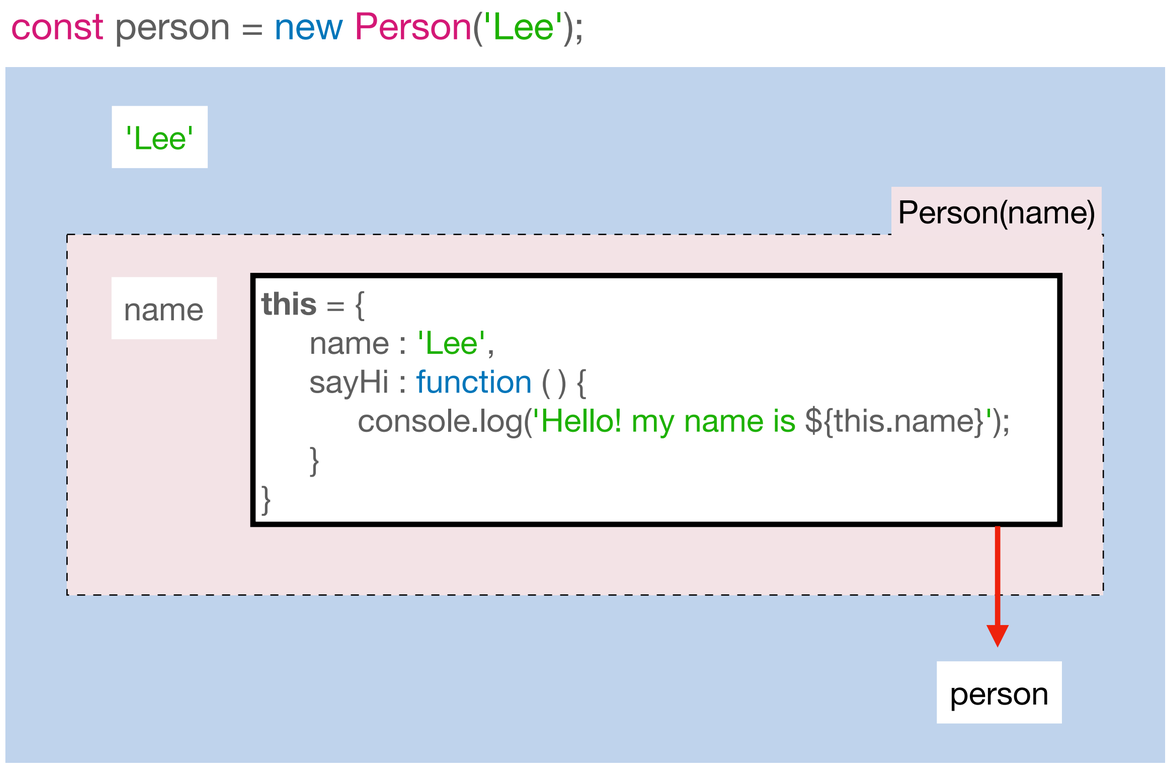



생성자 함수 (constructor)

생성자 함수 인스턴스 생성 과정

```
function Person(name) {
    // 런타임 이전에 암묵적으로 빈 객체가 생성되고 this에 연결된다
    this.name = name;
    this.sayHi = function ( ) {
        console.log('Hello! my name is ${this.name}');
    }
    // new 키워드로 호출된 생성자 함수는 암묵적으로 this를 반환한다
}

const person = new Person('Lee');
```

Done

reference : deepdive