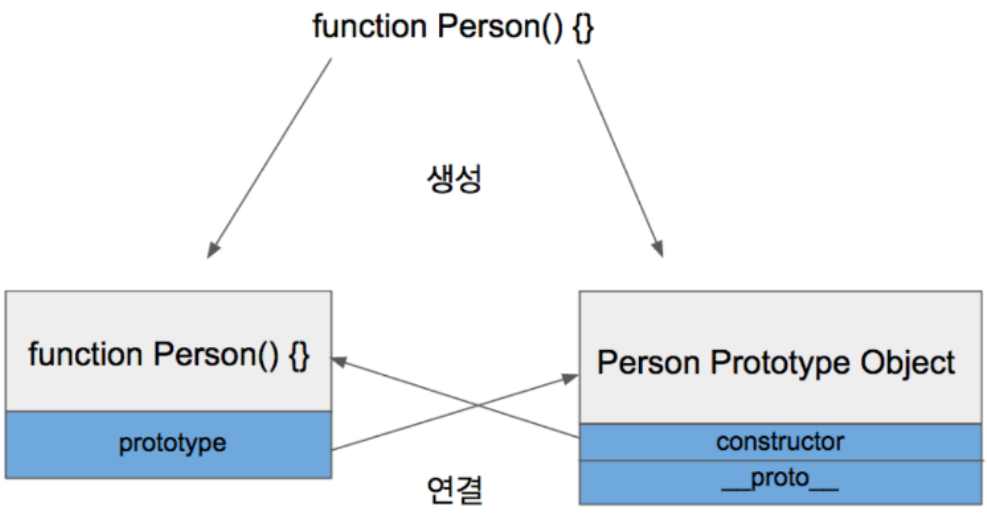
2023.03.12 Prototype

Prototype

1. Prototype 기반 언어
    a. 모든 객체들이 method와 property를 상속 받기 위해 Prototype 객체를 가진다는 의미입니다.
        i. 객체를 정의하면 해당 객체와 함께 **해당 객체의 Prototype 객체가 생성**됩니다.
        ii. 해당 객체는 **prototype 속성**을 가지며, 이는 함께 생성된 **Prototype 객체를 참조**합니다.
        iii. 함께 생성된 Prototype 객체 내부에는 **constructor 함수**가 존재하며 이는 **생성된 객체를 참조**합니다.
        iv. 또한, 함께 생성된 Prototype 객체 내부에는 **[[Prototype]]**가 존재하는데, 이는 method와 property를 상속해 준 **상위 객체**(원형)를 의미합니다.
            1. 해당 객체는 **[[Prototype]]**을 통해 상위 객체와 연결되어 있으므로, 상위 객체의 method나 property에 접근이 가능합니다.
            2. **__proto__**를 통해 **[[Prototype]]**에 직접 접근, 수정 등이 가능합니다. 그러나 이는 성능에 악영향을 미치므로 `Object.getPrototypeOf()` 와 같은 Static method를 이용하는 것이 좋습니다.



        v. 이렇게 모든 객체들이 Prototype을 통해 상위 객체들과 연결되어 있는 것을 **Prototype Chain**이라 부릅니다. 바꿔 말하면, 하위 객체에서도 사용할 수 있도록 **상속되는 method나 property는 해당 객체의 Prototype 객체에 정의**됩니다.
2. 작동 예시
    a. 객체 생성 시, 아래와 같이 작동합니다.

```
function Animal(name) {
  this.name = name;
}

console.log(Animal);
/* 해당 객체 생성
ƒ Animal(name) {
    this.name = name;
}
*/

console.log(Animal.prototype);
/* 해당 객체의 Prototype 객체 생성
{constructor: ƒ Animal(name)}
*/

console.log(Animal.prototype.__proto__);
console.log(Object.getPrototypeOf(Animal.prototype));
/* 해당 객체의 Prototype 객체에서 원형(여기서는 Object.prototype)에 접근 가능
constructor: ƒ Object()
hasOwnProperty: ƒ hasOwnProperty()
isPrototypeOf: ƒ isPrototypeOf()
propertyIsEnumerable: ƒ propertyIsEnumerable()
toLocaleString: ƒ toLocaleString()
toString: ƒ toString()
valueOf: ƒ valueOf()
__defineGetter__: ƒ __defineGetter__()
__defineSetter__: ƒ __defineSetter__()
__lookupGetter__: ƒ __lookupGetter__()
__lookupSetter__: ƒ __lookupSetter__()
__proto__: null
get __proto__: ƒ __proto__()
set __proto__: ƒ __proto__()
*/
```

b. `new` 키워드를 통한 instance 생성 시, 아래와 같이 작동합니다.

```
function Animal(name) {
  this.name = name;
}

const kim1 = new Animal('kim');

console.log(kim1)
/* kim1은 name이 'kim'으로 정해진 Animal 객체
Animal {name: 'kim'}
*/

console.log(kim1.__proto__)
console.log(Object.getPrototypeOf(kim1))
/* kim1의 원형은 Animal.prototype과 같음을 확인할 수 있음
{constructor: ƒ Animal(name)}
*/
```

- 즉, 객체의 instance는 엄밀하게 말하자면, 객체의 Prototype 객체로부터 method와 property를 상속 받아 만들어진 하위 객체라고 볼 수 있습니다.

c. 상위 객체의 method 호출 시, 아래와 같이 작동합니다.

```
function Animal(name) {
  this.name = name;
}

const kim1 = new Animal('kim');

console.log(kim1.hasOwnProperty())
/*
1. Animal {name: 'kim'}에서 hasOwnProperty method를 찾습니다.
2. 없으면 kim1의 [[Prototype]]인 Animal.prototype에서 hasOwnProperty method를 찾습니다.
3. 없으면 Animal.prototype의 [[Prototype]]인 Object.prototype에서 hasOwnProperty method를 찾습니다.
*/
```

- 위와 같이 Prototype Chain으로 인해 불필요한 메모리의 낭비를 줄일 수 있습니다.