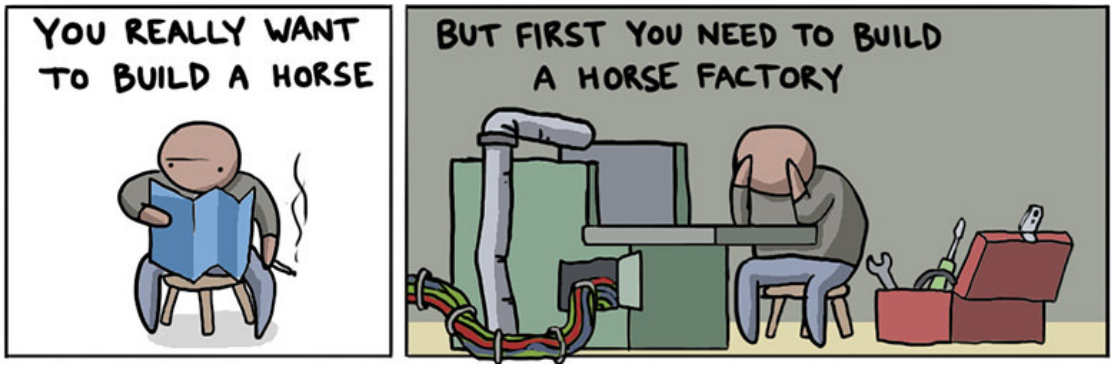
객체지향 프로그래밍

객체지향 프로그래밍(OOP)이란 프로그래밍하려는 대상을 하나의 객체(사물)로 정의하는 설계 방법으로 객체의 관점에서 구조를 만들고 사용하는 방법입니다. 조금 더 구체적인 설명을 덧 붙이자면 단순한 자료 구조(변수)를 넘어서 기능(메서드)을 포함한 형태로 객체를 사용하는 프로그래밍입니다.

왜 쓰나요?

- 각 기능을 독립적인 모듈로 관리할 수 있으며, 다른 사람이 내 코드의 내용을 직접 수정하지 않고 데이터에 접근하게 만들 수 있다. 따라서 코드 재사용성을 높이고 의존성을 관리하기 쉬워진다. 대신 코드 설계를 잘해야 한다.
- 대신 코드 설계를 잘해야 한다. 객체 사이의 관계를 생각하지 않고 무작정 코드를 작성하기 시작하면 모든 것이 꼬여버릴 수 있다.



Class

구조체와 함수를 합쳐 선언하는 것

```
// class
class Character {
  name = "young"
  #hp = 100
  #mp = 50

  attck() {...}
  useSkill() {... this.#mp -= 10; ... }
  moveTo(toX, toY) {...}
}
```

```
// object
var character = new Character();
character.name = "영민" // public한 필드는 외부에서 수정이 가능한 잠재적 위험이 있다!
```

외부로 노출해야 하는 값과 내부에서만 사용하는 값을 구분하여야 하는데
이때 필요한 메소드나 데이터들만 열어두는 특성을 **캡슐화** 라고 합니다.

- 이때 private 속성을 만들면 접근이 불가능해지는데 암묵적으로 변수명 앞에 `_`,`#` 를 붙여 만들거나
- 클로저를 사용하는 방법이 있다.

  ▼ 참고링크

  https://blog.shiren.dev/2016-06-27-클로저,-그리고-캡슐화와-은닉화/

상속

객체가 중심이 되어 그것을 재사용하는 것은 좋은데 객체는 여러 개의 변수와 여러 개의 함수가 섞여있어, 일부는 재사용을 하고 일부는 달라져야 하는 경우가 자주 생긴다는 것을 알게 됩니다.
객체의 일부분만 재사용하는 방법 ⇒ 상속

```
class Monster {
  name = ""
  hp = 0

  die() {...}
  attck() {...}
  moveTo(toX, toY) {...}
}

class Slime extends Monster {
  name = "슬라임"
  hp = 35

  attck() {...}
}

class RedPig extends Monster {
  name = "빨간돼지"
  hp = 70

  attck() {...}
}
```

이런식으로 사용이 가능합니다.

이 과정속에 공통적인 부분을 모아 상위의 개념으로 이름을 붙이는 것을 **추상화** 라고 합니다.
상속과 추상화를 이용하여 객체를 재사용 할 수 있습니다.

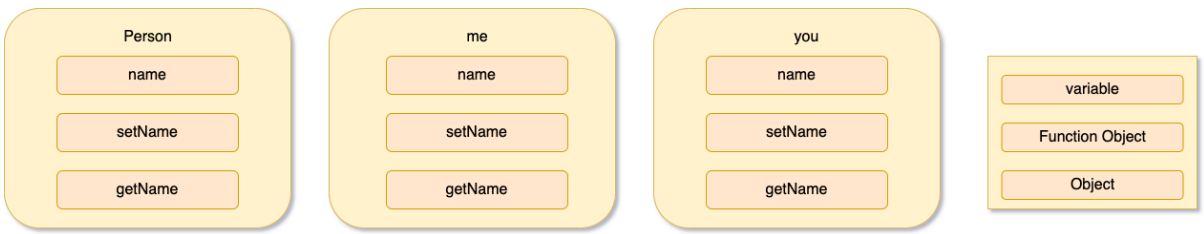
- 추상화 = Monster
- 상속 = Slime, RedPig

자바스크립트에서의 OOP

JS를 이용한 객체지향 프로그래밍 코드 예시)

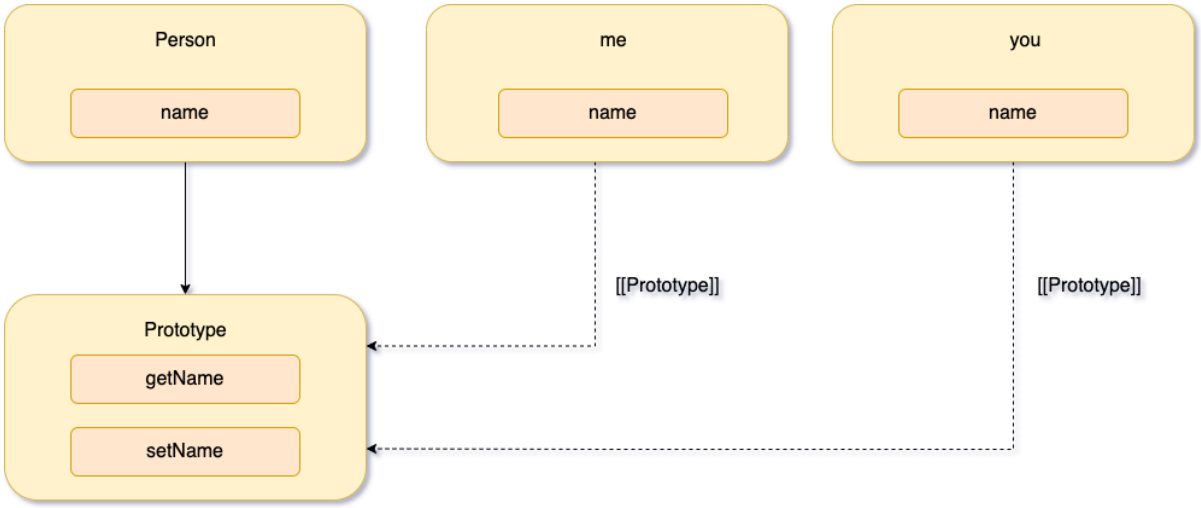
```
function Person(arg) {

    this.name = arg;

    this.getName = function() {
        return this.name;
    }

    this.setName = function(value) {
        this.name = value;
    }
}

var me = new Person("아바타");
console.log(me.getName());

me.setName("영민");
console.log(me.getName());

var me = new Person("me");
var you = new Person("you");
var him = new Person("him");
```

공통으로 사용할 수 있는 setName() 함수와 getName() 함수를 따로 생성하고 있습니다.
이는 불필요하게 메모리에 올리고 사용함으로 자원 낭비가 됩니다.



프로토 타입을 이용하여 직접 연결을 하는 방법

```
function Person(arg){
    this.name = arg;
}

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

Person.prototype.setName = function(value) {
    this.name = value;
}

var me = new Person("me");
var you = new Person("you");

console.log(me.getName());
console.log(you.getName());
```



protoType을 이용하여 객체를 정의하고 사용하는 코드

객체지향 프로그래밍은 객체를 정의하고 객체를 생성해서 만들어진 객체를 사용한다.

- protoType을 이용하여 객체를 정의
- new 생성자를 이용하여 객체를 만들고 . 을 이용하여 객체가 가지고 있는 메소드를 사용