Prototype

프로그래밍 언어의 종류에는 **객체지향 언어**와 **절차지향 언어**가 있다.

그 중 **객체지향 언어**는 **클래스 기반 객체지향**과 **프로토타입 기반 객체지향**으로 구분된다.

ES5까지의 Javascript는 **프로토타입 기반 객체지향 언어**에 해당한다.

#01. 객체의 의미

하나의 변수 안에 비슷한 특성을 갖는 변수와 함수가 내장된 형태.

객체 안에 내장된 변수를 **멤버변수** 혹은 **프로퍼티(속성)**라고 한다.

객체 안애 내장된 함수를 **메서드**라고 한다.

#02. 생성자 함수

1) 생성자 함수 정의하기

함수를 new 연산자를 사용하여 호출하는 경우 이 함수는 Javascript에 의해 객체를 만들기 위한 함수로 분류된다.

```
function helloworld() {
    ...
}

const h = new helloworld();
```

위와 같이 사용되는 함수를 **생성자(Constructor)**한다.

생성자를 호출하면서 리턴을 변수는 생성자 함수 자체의 리턴 유무에 상관 없이 객체가 된다.

화살표 함수 형식은 생성자로 사용할 수 없다.

2) 생성자에 멤버변수 포함시키기

생성자 함수 안에서 this 키워드를 통해 정의한 변수는 객체의 멤버변수 역할을 한다.

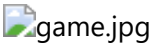
멤버변수는 객체 내부에 포함되는 형식으로 존재한다.

```
function helloworld() {
  this.x = 5;
  this.y = 10;
}

const h = new helloworld();
const z = h.x + h.y; // 5 + 10
```

같은 생성자를 통해 할당된 객체는 동일한 자료 구조를 갖지만 각각 다른 정보를 저장할 수 있다.

하나의 생성자를 통해 동일한 구조를 갖는 객체를 여러개 생성한 예

 game.jpg

3) 파라미터를 갖는 생성자

객체에 포함되는 멤버변수의 초기값을 설정하기 위한 용도로 생성자 함수는 파라미터를 갖을 수 있다.

생성자 파라미터를 통해 객체 생성시 초기값을 한번에 설정하면, 객체를 생성한 후 개별적으로 파라미터를 지정하는 것 보다 전체 코드가 축약된다.

#02. 메서드

객체에 포함된 함수.

특정한 목적을 위한 함수가 다수 존재할 때, 이 함수들을 그룹화 해 놓은 형태를 객체로 볼 수 있다.

1) prototype 속성

Javascript의 모든 객체는 `prototype`이라는 속성을 갖는다.

이 속성을 생성자 함수에 대해 활용하면 생성자 함수에 속한 다른 변수나 함수를 추가할 수 있다. (주로 함수 추가에 사용한다.)

즉, 먼저 정의된 생성자의 기능을 prototype을 통해 확장할 수 있다.

```
생성자함수.prototype.메서드이름 = function(파라미터1, 파라미터2, ... 파라미터n) {
    ...
    return 돌려줄_값;
}

// 메서드 호출하기
const 객체이름 = new 생성자함수();
const k = 객체이름.메서드이름();
```

2) 메서드 정의하기

`Object.defineProperty(생성자이름.prototype, 함수이름, {function 정의})` 형식으로 특정 멤버변수에 대한 `getter`, `setter`를 정의할 수 있다.

#03. getter, setter

객체지향에서는 객체를 통한 멤버변수의 직접 접근이 멤버변수에 값을 대입하는 과정에서 그 값의 적절성을 판단할 수 없고, 무조건적으로 대입하기 때문에 코드 보안에 부적절하다고 본다.

멤버변수에 값을 간접적으로 대입하는 특수한 형태의 함수를 `setter`, 멤버변수의 값을 리턴받기 위해 사용하는 특수한 형태의 함수를 `getter`라고 한다.

1) getter, setter 정의하기

`Object.defineProperty(생성자이름.prototype, 함수이름, {getter,setter 정의})` 형식으로 특정 멤버변수에 대한 `getter`, `setter`를 정의할 수 있다.

getter, setter는 같은 함수이름을 공유한다.

```
Object.defineProperty(생성자이름.prototype, "함수이름", {
    get: function() {
        ...
        return this.멤버변수
    },
    set: function(파라미터) {
        ...
        this.멤버변수 = 파라미터;
    }
});
```

2) getter, setter 활용하기

함수이지만 변수처럼 사용한다.

```
const 객체 = new 생성자이름();

// setter를 호출한다. 대입되는 값은 setter에 전달되는 파라미터.
객체.함수이름 = 000;

// getter를 호출한다. 멤버변수를 대입하는 것 같지만 실제로는 getter를 호출해서 리턴값을 받는 과정이다.
const 변수 = 객체.함수이름;
```

#04. JSON 구문 형식을 활용한 Prototype 정의

생성자이름.prototype = { ... } 형식으로 getter, setter, 메서드 등을 한번에 추가할 수 있다.

이러한 형식으로 생성자, 멤버변수, getter, setter, 메서드 등이 묶여 있는 단위를 **클래스**라 한다.

클래스에 정의된 기능을 하나의 변수에 모두 부여한 형태가 **객체**이다.

1) 클래스 완전체 정의하기

생성자 정의

생성자는 익명함수 형식으로 정의할 수 도 있다.

```
function 생성자이름(파라미터1, 파라미터2, ... 파라미터n) {
    this.멤버변수1 = 파라미터1;
    this.멤버변수2 = 파라미터2;
    ...
    this.멤버변수n = 파라미터n;
}
```

JSON을 활용하여 getter, setter와 메서드 추가하기

```
생성자이름.prototype = {
    // getter와 setter의 이름은 동일해야 하고, getter/setter 쌍이 멤버변수 수 만큼 정의된다.
    get getter이름() {
        return this.멤버변수;
    },

    set setter이름(파라미터) {
        this.멤버변수 = 파라미터;
    },

    메서드1이름 : function() {
        ...
    },

    메서드n이름 : function() {
        ...
    }
};
```

정의된 클래스를 통해 객체 생성하기

```
const 객체 = new 생성자이름();
```

좀 더 정확하게 구분하자면

`const 객체`로 선언된 객체는 **객체 참조 변수**라고 부르고, `new 생성자이름()` 부분에서 생성된 object를 인스턴스 라고 한다.