10장 객체 리터럴

객체란?

- JS 객체기반의 프로그래밍 언어이며, JS를 구성하는 거의 모든것이 객체임
    - 원시 값을 제외한 나머지 값 모두를 말함
    - 원시 값은 immutable value이지만 객체 값 mutable value

```
var count = {
  num : 0,  // property key : property value (property)
  increase : function() { // method
    this.num++;
  }
}
```

- 프로퍼티
    - 객체의 상태를 나타내는 값
- 메서드
    - 프로퍼티(상태 데이터)
- JS 객체와 함수
    - 함수로 객체를 생성하기도 하며 함수 자체가 객체이기도 함

객체 리터럴에 의한 객체 생성

- 인스턴스
  - 클래스에 의해 생성되어 메모리에 저장된 실체, 객체가 메모리에 저장되어 실제로 존재하는 것에 초점을 맞춘 용어
  - OOP에서 객체는 클래스와 인스턴스를 포함한 개념이며 클래스는 인스턴스를 생성하기 위한 템플릿의 역할을 함
- JS 는 prototye 기반 객체지향 언어로 class 기반 객체지향 언어와 달리 다양한 객체 생성 방법을 지원
  1. 객체 리터럴
  2. Object 생성자 함수
  3. 생성자 함수
  4. Object.create 메서드
  5. 클래스(E6)

```
var turtle = {
  name : 'tur',
  sayHelloToMyLittleFriend : function() {
    console.log(`${name} Bang!`);
  }
};

console.log(typeof turtle); // object
console.log(turtle); // {name:'tur',sayHelloToMyLittleFriend:f}

// ===================================================================

var empty = {}: // 빈 객체
console.log(typeof empty); // object
```

- 객체 리터럴은 { … } 내에 0개 이상의 프로퍼티를 정의하며 변수에 할당되는 시점에 JS 엔진은 객체 리터럴을 해석해 객체를 생성함
  - 만약 중괄호 내에 프로퍼티를 정의하지 않으면 빈 객체가 생성됨
  - 이 때 객체 리터럴의 중괄호는 코드 블록을 의미하지 않아서 객체 리터럴을 닫는 중괄호 뒤에는 세미콜론을 붙임

프로퍼티

- **객체는 프로퍼티의 집합이며, 프로퍼티는 키와 값으로 구성됨**

```
var person = {
  name : 'turtle',
  age : 23
};
```

- 프로퍼티 키 - 빈 문자열을 포함하는 모든 문자열 또는 심벌 값
  - 키 값은 식별자 네이밍 규칙을 따라야 하는 것은 아님, 다만 네이밍 규칙을 따르는 키와 그렇지 않은 프로퍼티 키는 미묘한 차이가 존재함
  - 심벌 값도 프로퍼티 키로 사용가능하지만 일반적으로 string을 사용함
  - 보통 프로퍼티 값은 식별자 네이밍 규칙을 지키는 이름인 경우를 제외하면 ‘’ 또는 “”를 키 값에 씌워야 함
- 프로퍼티 값 - JS에서 사용할 수 있는 모든 값

아래의 코드를 실행해보시고 왜 결과 값이 그렇게 나오는지 생각해보세요

```
var person = {
  firstName : 'turtle',
  last-name : 'park'
};

console.log(person);
//===============================================

var word1 = {
  var: '',
  function: ''
};

console.log(word1);
//===============================================

//프로퍼티 키 동적 생성
var objES5 = {}
var keyES5 = 'ES5'
objES5[keyES5] = 'world';

console.log(objES5);
//===============================================

//계산된 프로퍼티 이름
var keyES6 = 'HELL';
var objES6 = {[keyES6]: 'o'};

console.log(objES6);
//===============================================

var emptyObj = {
'' : ''
};
console.log(emptyObj);
//===============================================

var numObj = {
1 : 0,
2 : 1,
3 : 2
};
console.log(numObj);
//===============================================

var duplicateObj = {
  name : 'park',
  name : 'kim'
};
console.log(duplicateObj);
//===============================================
```

메서드

- JS에서 사용할 수 있는 모든 값은 프로퍼티 값으로 사용할 수 있음
  - 함수는 일급 객체라서 값으로 취급되어 프로퍼티 값으로 사용할 수 있음
  - 프로퍼티 값이 함수일 경우 일반 함수와 구분하기 위해 메서드라 부름

```
var circle = {
  radius: 5,
  getDiameter: function(){
    return 2 * this.radius;
  }
};

console.log(circle.getDiameter());
```

프로퍼티 접근

1. 마침표 프로퍼티 접근 연산자 (.)를 이용한 방법
2. 대괄호 프로퍼티 접근 연산자 ([ ])를 이용한 방법
   - 해당 방법을 이용시 대괄호 안의 프로퍼티 키 네임을 무조건 ‘’ 을 감싸야함
   - 해당 방법에서 ‘’ 가 없는 경우는 키 값이 문자열이 아닌 숫자형일때는 생략가능함
   - 만약 ‘’ 없이 사용한다면 선언된 키는 찾았지만 찾지 못해서 ReferenceError가 나옴
   - 만약 객체에 존재하지 않는 프로퍼티에 접근하면 undefined를 반환함 → Error 발생 X
- JS에서 사용 가ㅣ능한 유효한 이름이면 마침표와 대괄호 모두 사용 가능함

```
var person = {
  name : 'park'
}

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

브라우저 환경과 Nodejs 환경을 준비하고 아래의 코드를 돌려봅시다.

```
var wind = {
  'last-name' : 'park',
  1: 10
};

wind.'last-name';
wind.last-name;

wind[last-name];
wind['last-name'];

wind.1;
wind.'1';
wind[1];
wind['1']
```

프로퍼티 값 갱신

- 존재하는 프로퍼티에 값을 할당하면 갱신됨

```
var windAndWish = {
  wow: 'yayaya'
};

windAndWish.wow = 'everybodySay';

console.log(windAndWish);
```

프로퍼티 동적 생성

- 존재하지 않은 프로퍼티에 값을 할당하면 프로퍼티가 동적으로 생성되어 추가되고 프로퍼티 값이 할당됨

```
var person = {
 beautiful : 'pain'
};

person.time = 'sad';

console.log(person);
```

프로퍼티 삭제

- delete 연산자를 이용해서 삭제함
    - 만약 삭제할 연산자가 없으면 에러 없이 무시됨

```
var aaaaa = {
  name : 'park',
  age : 25
};

delete aaaaa.age;
delete aaaaa.sing; // 해당 프로퍼티가 없으니 무시됨

console.log(aaaaa);
```

ES6에서 추가된 객체 리터럴 확장 기능

프로퍼티 축약 표현

- ES6 기준으로 프로퍼티 값으로 변수를 사용하는 경우 변수 이름과 프로퍼티 키가 동일한 이름일 때 프로퍼티 키를 샐략 가능함
    - 프로퍼티 키는 변수 이름으로 자동 생성됨

```
let x =1, y=2;
const obj = {x,y};

console.log(obj);
```

계산된 프로퍼티 이름

- 문자열 또는 문자열로 타입 변환할 수 있는 값으로 평가되는 표현식을 사용해 프로퍼티 키를 동적으로 생성 할 수도 있음
    - 다만 해당 키를 [key] 형식으로 묶어놔야함

```
const prefix = 'props';
let i = 0;

const obj = {
  [`${prefix}- ${++i}`]: i,
  [`${prefix}- ${++i}`]: i,
  [`${prefix}- ${++i}`]: i
```

```
};

console.log(obj);
```

메서드 축약 표현

- ES6에서는 메서드 정의할 때 function 키워드 생략 가능

```
const obj = {
  song : '하루가 다 지났어~',
  sayOh() {
  console.log('너를 그리워하다~' + this.song)
  }
};

obj.sayOh();
```