삼육대학교 SW융합교육원 김진호

자바스크립트 기초교육

<class>
</body>
<head>
<title>

04. object literal (리터럴 객체)

4-1. object

4-2. property

4-3. ES6

4-4. additional-operator-and-traversal

4. object literal

4-1. object (객체)

자바스크립트는 객체 기반 프로그래밍 언어로 원시 값을 제외한 나머지 값(함수, 배열, 정규 표현식 등)은 모두 객체이다.
객체는 0개 이상의 프로퍼티로 구성된 집합이다.
프로퍼티는 키(key)와 값(value)으로 구성된다. 자바스크립트에서 사용할 수 있는 모든 값은 프로퍼티 값이 될 수 있다.
프로퍼티 값이 함수일 경우 method라고 부른다.

4. object literal

4-1. object (객체)

객체 생성 방법

자바, C++ 같은 클래스 기반 객체 지향 언어는 클래스를 사전에 정의하고 필요한 시점에 new 연산자와 함께 생성자를 호출하여 인스턴스를 생성하는 방식으로 객체를 생성한다. 자바스크립트는 프로토타입 기반 객체지향 언어로 클래스 기반 객체 지향 언어와는 달리 다양한 객체 생성 방법 지원한다.

1. 객체 리터럴
2. Object 생성자 함수
3. 생성자 함수
4. Object.create 메서드
5. Class(ES6)

이 중 가장 일반적이고 간단한 방법이 객체 리터럴을 사용하는 방법이다. 중괄호 내 0개 이상의 프로퍼티를 정의한다.

4. object literal

4-1. object (객체)

객체 생성 방법

```
var student = {
    // 키-값 쌍으로 구성 된 프로퍼티
    // 프로퍼티 : 객체의 상태를 나타내는 값(data)
    name : '유관순',
    age : 16,
    // 메서드 : 프로퍼티(상태 데이터)를 참조하고 조작할 수 있는 동작(behavior)
    getInfo : function(){
        return `${this.name}(은)는 ${this.age}세입니다.`;
    }
};
```

- 객체 리터럴의 중괄호는 코드 블록을 의미하지 않는다. 따라서 닫는 중괄호 뒤에는 세미콜론을 붙인다. 숫자 값이나 문자열을 만드는 것과 유사하게 리터럴로 객체를 생성한다. 객체 리터럴에 프로퍼티를 포함시켜 객체를 생성함과 동시에 프로퍼티를 만들 수도 있고, 객체를 생성한 이후에 프로퍼티를 동적으로 추가할 수도 있다. 객체 리터럴 외의 객체 생성 방식은 모두 함수를 사용해서 생성하므로 함수 학습 이후에 다시 학습할 것이다.

4. object literal

4-2. property (속성)

객체는 프로퍼티의 집합이며, 프로퍼티는 키와 값으로 구성 된다.

4-2-1. property

```
var student = {
    // 프로퍼티 키는 name, 프로퍼티 값은 '유관순'
    name : '유관순',
    // 프로퍼티 키는 age, 프로퍼티 값은 16
    age : 16,
    // 프로퍼티 나열은 쉼표로 구분하며 마지막 프로퍼티 뒤에 쉼표를 사용해도 된다.
};
```

- **프로퍼티 키 : 빈 문자열을 포함하는 모든 문자열 또는 심벌 값**
  - 프로퍼티 값에 접근하기 위한 식별자
  - 문자열이므로 따옴표를 사용하지만 식별자 네이밍 규칙을 따르는 경우 사용하지 않아도 된다.
  - 단, 식별자 네이밍 규칙을 따르지 않는 이름은 따옴표를 반드시 사용해야 하며 가능한 식별자 네이밍 규칙을 따르는 것을 권장한다.
- **프로퍼티 값 : 자바스크립트에서 사용할 수 있는 모든 값**

4. object literal

4-2. property (속성)

4-2-2. method

```
var dog = {
    name : '뽀삐',
    // 메서드 - 객체에 묶여 있는 함수
    eat : function (food) {
        console.log(`${this.name}(은)는 ${food}를 맛있게 먹어요.`);
    }
}
```

- 자바스크립트의 함수는 객체이다.
  함수는 값으로 취급할 수 있고 프로퍼티 값으로 사용할 수 있다.

4. object literal

4-2. property (속성)

4-2-3. property accessor (프로퍼티 접근)

```
console.log(dog.name);          // 마침표 표기법(dot notation)
console.log(dog['name']);       // 대괄호 표기법(square bracket notation)
```

- 마침표 표기법(dot notation)
- 대괄호 표기법(square bracket notation)
  - 프로퍼티 키는 반드시 따옴표로 감싼 문자열 사용한다.
  - 프로퍼티 키가 식별자 네이밍 규칙을 준수하지 않는 이름일 경우 반드시 대괄호 표기법 사용한다.
  - 프로퍼티 키가 숫자로 이뤄진 문자열인 경우 따옴표를 생략할 수 있다.

4. object literal

4-2. property (속성)

4-2-4. property change, add, remove (프로퍼티 값 변경, 추가, 삭제)

```
var dog = { name : '뽀삐' }
dog.name = '두부';              // 프로퍼티 값 갱신
dog.age = 3;                    // 프로퍼티 값 추가
delete dog.age;                 // 프로퍼티 삭제
```

- **이미 존재하는 프로퍼티에 값을 할당하면 프로퍼티 값이 갱신된다.**
- **프로퍼티 동적 추가**
  - 존재하지 않는 프로퍼티에 값을 할당하면 프로퍼티가 동적으로 생성 되어 추가되고 프로퍼티 값이 할당 된다.
- **프로퍼티 삭제**
  - delete 연산자는 객체의 프로퍼티를 삭제한다.
  - 만약 존재하지 않는 프로퍼티를 삭제하면 아무런 에러 없이 무시된다.

4. object literal

4-3. ES6

4-3-1. property value shorthand (프로퍼티 값 단축 구문)

```
var id = 'p-0001';
var price = 30000;
var product2 = { id, price };
```

ES6에서는 프로퍼티 값으로 변수를 사용하는 경우 변수 이름과 프로퍼티 키가 동일한 이름일 때 프로퍼티 키를 생략할 수 있다. 프로퍼티 키는 변수 이름으로 자동 생성 된다.

4. object literal

4-3. ES6

4-3-2. computed property (계산된 프로퍼티 이름)

```
var prefix = 'key';
var index = 0;
var obj = {
    [`${prefix}-${index++}`] : index,
    [`${prefix}-${index++}`] : index,
    [`${prefix}-${index++}`] : index
};
```

- ES5에서 계산된 프로퍼티 이름으로 프로퍼티 키를 동적 생성하려면 객체 리터럴 외부에서 대괄호 표기법을 사용해야 한다.
- ES6에서는 객체 리터럴 내부에서도 계산된 프로퍼티 이름으로 프로퍼티 키를 동적 생성할 수 있다.

4. object literal

4-3. ES6

4-3-3. method shorthand (메서드 단축)

```
var dog = {
    name : '두부',
    // 메서드 축약 표현
    eat(food) {
        console.log(`${this.name}(은)는 ${food}를 맛있게 먹어요.`);
    }
}
```

- ES5에서 메서드를 정의하려면 프로퍼티 값으로 함수를 할당한다.
- ES6에서는 메서드를 정의할 때 function 키워드를 생략한 축약 표현을 사용할 수 있다.

4. object literal

4-4. additional-operator-and-traversal (추가 연산자와 순회)

4-4-1. in operator (in 연산자)

```
var student = {
    name : '유관순'
};
console.log("name" in student);              // true - 존재
console.log("height" in student);            // false - 존재하지 않음
```

- in 연산자를 이용해 프로퍼티 존재 여부를 확인할 수 있다.

4. object literal

4-4. additional-operator-and-traversal (추가 연산자와 순회)

4-4-2. for in

```
var student = {
    name : '유관순',
    age : 16
};

for (var key in student) {
    console.log(`${key} : ${student[key]}`);
   // name : '유관순', age : 16 출력
}
```

- for in 반복문을 이용해 객체의 모든 키를 순회할 수 있다.