JavaScript Review

var, let, const

선언방법 - ES6

ECMAScript 6

var

var 변수

```
//var.
var 변수 = 값;
```

var 변수

```
var foo = 1;
```

var 변수

```
var foo = 'apple';
```

var 변수

```
var foo = 1;
console.log(typeof foo);
```

var 변수

```
var foo = 'apple';
console.log(typeof foo);
```

```
var foo = {name : 'apple', number : 1};
console.log(typeof foo);
```

```
var foo = ['apple', 'orange', 'cherry'];
console.log(typeof foo);
```

```
var foo = [1,2,3,4];
console.log(typeof foo);
```

let

let 상수

```
//let.
let 상수 = 값;
```

let 상수

```
var foo = 1;
foo = 2;
console.log(foo);
```

let 상수

```
let foo = 1;
foo = 2;
console.log(foo);
```

```
var foo = 'apple';
console.log('1', foo);
if(true){
    var foo = 'orange';
    console.log('2', foo);
}
console.log('3', foo);
```

let 상수

```
let foo = 'apple';
console.log('1', foo);
if(true){
    let foo = 'orange';
    console.log('2', foo);
}
console.log('3', foo);
```

let 상수

```
let foo = 'apple';
console.log('1', foo);
if(true){
    console.log('2', foo);
    foo = 'orange';
    console.log('3', foo);
}
console.log('4', foo);
```

```
let foo = 'apple';
console.log('1', foo);
if(true){
    console.log('2', foo);
    let foo = 'orange';
}
console.log('3', foo);
```

const

const 상수선언

```
//const.
const 상수선언 = 값;
```

const 상수선언

```
const foo = 1;
foo = 1;
```

const 상수선언

```
const foo = [1,2];
const bar = foo;

foo.push(3);
bar[0] = 10;

console.log('1', foo);
console.log('2', bar);
```

ES6 에서 var 지양하고, let, const 를 사용한다.

원시형에서 변수는 let, 상수는 const 로 선언한다.

참조형은 const 로 선언한다.

Object

변형 가능한 속성의 집합

배열, 함수, 정규표현식 등 객체

클래스가 없다. (class-free)

새로운 속성의 이름이나 값에 어떠한 제약사항도 없다.

객체는 또 다른 객체를 포함 가능하다.

Prototype 연결 특성

Object 객체

```
//Object.
var object = new Object();
var object = {};
```

Object 객체

```
//Object.
var object = new Object();
var object = {};
```

Object :: Property

Object 객체

```
var car = {};
car.name = 'Tesla';
car.model = 'Model 3';
car.color = 'white';
```

Object 객체

```
var car = {
    name : 'Tesla',
    model : 'Model 3',
    color : 'white'
};
```

Object :: hasOwnProperty

```
var car = {};
car.tesla = 'Model 3';
car.bmw = '';
car.audi = '';
console.log(car.hasOwnProperty('tesla'));
```

Object :: Property 생성

Object 객체

```
var car = {};
car.name = 'Tesla';
car['name'] = 'Tesla';
console.log(car.name);
console.log(car['name']);
```

Object :: Property 삭제

Object 객체

```
var car = {};
car.name = 'Tesla';
delete car.name;
console.log(car.name);
```

Object :: Method

```
var car = {};
car.drive = function(){
    console.log('Car driving');
};
car.drive();
```

```
var car = {
    drive : function(){
        console.log('Car driving');
    }
};
car.drive();
```

Array

Array 배열

연관된 데이터를 모아서 관리하기 위해 사용하는 데이터 타입

변수의 집합, 데이터

여러 개의 데이터를 하나의 변수에 담아 관리 가능하다.

콤마(,)로 구분하여 나열된다.

복수의 데이터를 효율적으로 관리, 전달하기 위한 목적을 가진다.

데이터의 추가/수정/삭제가 편리하다.

Array 배열

```
//Array.
var array = new Array();
var array = [];
```

Array 배열

```
//Array.
var array = new Array();
var array = [];
```

Array 배열

```
var cars = ['Tesla', 'Audi', 'Volvo'];
cars.push('BMW');
cars[4] = 'Benz';
```

```
var cars = ['Tesla', 'Audi', 'Volvo'];
delete cars[2];
console.log(cars);
```

Array :: length

Array 배열

```
var cars = [];
cars.push('Tesla');
cars.push('Audi');
console.log(cars.length);
```

Array :: push

Array 배열

```
var cars = [];
cars.push('Tesla');
console.log(cars);
```

Array :: join

Array 배열

```
var cars = ['Tesla', 'Audi', 'Volvo'];
var message = cars.join();
console.log(message);
```

Array :: reverse

```
var cars = ['Tesla', 'Audi', 'Volvo'];
var reverse = cars.reverse();
console.log(reverse);
```

Loop

동일한 동작을 반복 수행

for, while, for...in, break, continue, forEach, for...of

컴퓨터에게 반복적인 작업을 지시하는 방법이다.

반복의 제한 조건이 있어야만 한다.

조건 형식이 없을 경우, 무한루프를 발생시킨다.

Loop :: for

Loop :: for

```
//for.
for(초기화; 반복조건; 반복시 실행되는 코드){
    반복해서 실행되는 코드
}
```

Loop :: for

```
for(var i = 0; i < 10; i++){
    document.write('for loop ' + i + '<br/>');
}
```

Loop :: for

```
var buttons = document.getElementsByClassName('button');
for(var i = 0, max = buttons.length; i < max; i++){
    buttons[i].innerHTML = 'Button ' + i;
}
```

Loop :: while

Loop :: while

```
//while.
while(조건){
    반복해서 실행되는 코드
}
```

Loop :: while

```
var count = 0;
var sum = 0;
while(count < 10){
    count++;
    sum += count;
}
```

Loop :: for 중첩

Loop :: for 중첩

```
//for 중첩.
for(var i = 0; i < 10; i++){
    for(var j = 0; j < 10; j++){
        document.write(String(i) + ' '+ String(j) + '<br/>');
    }
}
```

Loop :: break

Loop :: break

```
//break.
for(var i = 0; i < 10; i++){
    if(i === 5){
        break;
    }
    document.write('for loop ' + i + '<br/>');
}
```

Loop :: continue

Loop :: continue

```
//continue.
for(var i = 0; i < 10; i++){
    if(i === 5){
        continue;
    }
    document.write('for loop ' + i + '<br/>');
}
```

Loop :: for…in

Loop :: for...in

```
//for ... in.
for(변수 in 객체){
    반복해서 실행되는 코드
}
```

Loop :: for...in

```
var cars = {Tesla : '테슬라', Audi : '아우디'};
var name = '';
var nameKo = [];
for(let name in cars){
    nameKo.push(cars[name]);
}
```

Loop :: forEach

Loop :: forEach

```
//forEach
배열.forEach(function(변수){
    반복해서 실행되는 코드
});
```

Loop :: forEach

```
var cars = ['Tesla', 'Audi', 'Volvo', 'Benz'];
cars.forEach(function(name){
    console.log(name);
});
```

Loop :: forEach

```
var cars = ['Tesla', 'Audi', 'Volvo', 'Benz'];
cars.forEach(function(name, index){
    if(index >= 2){
        break;
    }
    console.log('name', name);
    console.log('index', index);
});
```

Loop :: for...of

Loop :: for...of

```
//for ... of
for(변수 of 배열){
    반복해서 실행되는 코드
}
```

Loop :: for...of

```
var cars = ['Tesla', 'Audi', 'Volvo', 'Benz'];
for(let name of cars){
    console.log(name);
}
```

Loop :: for...of

```
var cars = ['Tesla', 'Audi', 'Volvo', 'Benz'];
var index = 0;
for(let name of cars){
    if(index >= 2){
        break;
    }
    console.log(name);
    index++;
}
```

for...in - 객체

for...of - 배열

Function

실행문 집합을 포함한 객체

객체의 메서드와는 의미가 다르다.

외부 또는 내부 코드로부터 호출되는 하위 프로그램

first-class object

모든 함수는 function 객체

정보의 구성 및 은닉 등에 활용 - 클로저

모듈화의 기본 요소

```
//함수.
function name(){
    실행되는 코드
}
```

Function :: literal

```
var drive = function(){
    console.log('Car driving');
};
drive();
```

Function :: declaration

```
function drive(){
    console.log('Car driving');
};
drive();
```

Function :: return

```
function myCar(){
    var car = 'Tesla';
};
console.log(myCar());
```

```
function myCar(){
    var car = 'Tesla';
    return car;
};
console.log(myCar());
```

```
function myCar(){
    var car = 'Tesla';
    return car;
    car += ' Model 3';
};
console.log(myCar());
```

Function :: parameters

```
var coffee = '아메리카노';
function myCoffee(coffee){
    coffee = '에스프레소';
}
myCoffee(coffee);
console.log(coffee);
```

```
function orderCoffee(name, menu, cups){
    return name + ' 고객님, 주문하신 ' + menu + ', ' + cups + '잔 나왔습니다.';
}
var coffee = orderCoffee('김용원', '아메리카노', 3, '감사합니다.');
console.log(coffee);
```

Function :: Immediately-invoked function expression(IIFE)

```
var run = (function(){
    var x = 2;
    var y = 1;
    return x + y;
}());
console.log(run);
```

Function :: Scope

```
var name = 'Tesla';
function changeName(){
    var name = 'Audi';
}
changeName();
console.log(name);
```

```
var name = 'Tesla';
function changeName(){
    name = 'Audi';
}
changeName();
console.log(name);
```

```
function changeName(){
    var name = 'Tesla';
}
changeName();
console.log(name);
```