

INTERACTION PROGRAMMING 1

인터랙션 프로그래밍 1

4 Week.

2020. 4. 23.

JavaScript Review

Number

//정수.

1+1; 딱 떨어지는 숫자.

//실수.

1.5+1.5; 소수점이 있는 수, 현실을 반영한 수

//정수.

1+1; 딱 떨어지는 숫자.

//실수.

1.5+1.5; 소수점이 있는 수, 현실을 반영한 수

JavaScript 에서는 정수 / 실수 구분이 중요하지 않음.

다른 언어 (C, JAVA...) 에서는 중요함.

Operator

Operator 연산자

+ - * / %

Operator 연산자

+ - * / %

1 + 1;

10 - 1;

2 * 2;

9 / 3;

8 % 2;

String

String 문자

.....

String 문자

```
"사이에 작성";  
'사이에 작성';  
"사이에 작성";
```

//escape.

"사이에 \'작성\"'; 원래 가지고 있던 임무에서 탈출
'사이에 \"작성\\\"';

String 문자

```
//문자열 줄바꿈.  
"hello\nworld";
```

String 문자

1

"1"

String 의 연산

String 의 연산

```
"hello" + "world";  
"hello" + " world";
```

String 의 연산

1 + 1

"1" + "1"

1 + "1"

"hello world".length; 문자의 길이
"hello".indexOf('h'); 문자의 순번

Variable

Variable 변수

```
var variable;
```

Variable 변수

```
var variable = value;
```

변수명

Variable 변수

var variable = value;

변수명

변수값

Variable 변수

```
var a = 1;  
var b = 2;  
a + b
```

Variable 변수

```
var a = 1;  
var b = "2";  
a + b
```

Variable 변수

```
var a = "hello";
var b = "world";
a + b
```

```
var a = "hello";  
a = "HELL0~";
```

var 선언 이후에는 **var** 를 사용하지 않아도 된다.

```
var a = "hello";
a = "HELLO~";
a = a + " WORLD!";
a += ":^)";
```

변수는 무수히 많은 재활용이 가능하다.

100 에 10 을 더한 후, 10 으로 나누고, 10 을 뺀 후, 10 을 곱한다.

100에 10을 더한 후, 10으로 나누고, 10을 뺀 후, 10을 곱한다.

```
var sum = ( ((100 + 10) / 10 ) - 10 ) * 10;
```

100 에 10 을 더한 후, 10 으로 나누고, 10 을 뺀 후, 10 을 곱한다.

200 에 20 을 더한 후, 20 으로 나누고, 20 을 뺀 후, 20 을 곱한다.

300 에 20 을 더한 후, 10 으로 나누고, 30 을 뺀 후, 40 을 곱한다.

100에 10을 더한 후, 10으로 나누고, 10을 뺀 후, 10을 곱한다.

200에 20을 더한 후, 20으로 나누고, 20을 뺀 후, 20을 곱한다.

300에 20을 더한 후, 10으로 나누고, 30을 뺀 후, 40을 곱한다.

```
var a = 100;
```

```
var b = 10;
```

```
var sum = ( (a + b) / b ) - b ) * b;
```

100에 10을 더한 후, 10으로 나누고, 10을 뺀 후, 10을 곱한다.

200에 20을 더한 후, 20으로 나누고, 20을 뺀 후, 20을 곱한다.

300에 20을 더한 후, 10으로 나누고, 30을 뺀 후, 40을 곱한다.

```
var a = 200;
```

```
var b = 20;
```

```
var sum = ( (a + b) / b ) - b ) * b;
```

100에 10을 더한 후, 10으로 나누고, 10을 뺀 후, 10을 곱한다.

200에 20을 더한 후, 20으로 나누고, 20을 뺀 후, 20을 곱한다.

300에 20을 더한 후, 10으로 나누고, 30을 뺀 후, 40을 곱한다.

```
var a = 300, b = 20, c = 10, d = 30, e = 40;
```

```
var sum = ( (a + b) / c ) - d ) * e;
```

100에 10을 더한 후, 10으로 나누고, 10을 뺀 후, 10을 곱한다.

200에 20을 더한 후, 20으로 나누고, 20을 뺀 후, 20을 곱한다.

300에 20을 더한 후, 10으로 나누고, 30을 뺀 후, 40을 곱한다.

```
var a = 1000, b = 300, c = 5, d = 10, e = 60;
```

```
var sum = ( (a + b) / c ) - d ) * e;
```

변할 수 있는 영역과 변하지 않는 영역으로 구분할 수 있다.

(유지보수)

Data Type

Data Type

//Data type.

Boolean

Number

String

undefined

null

1

1 이 아닌 수

NaN

//Data type.

Boolean true | false

Number -1 0 1 2 3 4 5...

String "a" "b" "c"...

undefined undefined

null null

1 Boolean 의 true 로 간주

1 이 아닌 수 Boolean 의 false 로 간주

NaN 성립이 되지 않는 수, 계산할 수 없음을 의미함.

typeof

`typeof`

`typeof 1`

`typeof "1"`

`typeof []`

`typeof {}`

typeof

typeof 1	Number
typeof "1"	String
typeof []	Array
typeof {}	Object

undefined, null

`undefined`, `null`

`var a = null;` 값이 없는 상태, 의도해서 값이 없는 상태로 만든 것
`var a;` 값이 정의되지 않은 상태

비교 연산자

//연산자

+ - * / %

비교 연산자

//연산자
+ - * / %

//대입 연산자(이항 연산자)
var variable = value;

비교 연산자

```
//연산자  
+ - * / %
```

```
//대입 연산자(이항 연산자)  
var variable = value;
```

```
//비교 연산자  
Boolean  
true / false;  
1 / 0
```

비교 연산자

`==`

`> <`

`>= <=`

비교 연산자

$=$

$> <$

$\geq \leq$

값이 같은지 큰지 작은지를 비교

비교 연산자

`==`

`> <`

`>= <=`

`=>`

주의합니다. 다른 명령어입니다.

비교 연산자

```
var a = 1;  
var b = 1;  
a == b
```

비교 연산자

```
var a = 2;  
var b = 1;  
a > b
```

```
var a = 2;  
var b = 1;  
a < b
```

```
var a = 2;  
var b = 2;  
a >= b
```

```
var a = 1;  
var b = 2;  
a <= b
```

동등 연산자

동등 연산자

//대입 연산자(이항 연산자)

=

//동등 연산자

==

동등 연산자

```
var a = 1;  
var b = 2;  
a == b
```

동등 연산자

```
var a = 1;  
var b = 1;  
a == b
```

동등 연산자

```
var a = "one";
var b = "하나";
a === b
```

```
var a = "one";
var b = "one";
a === b
```

동등 연산자

```
var a = 1;  
var b = "1";  
a == b
```

동등 연산자

```
var a = 1;  
var b? = "1";  
a == b
```

동등 연산자

```
var a = 1;  
var b = "1";  
a === b
```

일치 연산자

일치 연산자

//대입 연산자(이항 연산자)

=

//동등 연산자

==

//일치연산자

====

```
var a = 1;          var a = 1;  
var b = "1";        var b = "1";  
a == b            a === b
```

정확히 일치 하는지를 비교, **Strict** (엄격한)
동등 연산자는 버그를 발생시킬 위험이 있다.

```
var a = null;  
var b;  
a == b
```

```
var a = null;  
var b;  
a == b
```

```
var a = null;  
var b;  
a === b
```

일치 연산자

$\theta \equiv -\theta$

```
true == 1
```

`true == 1`

`true === 1`

NaN === NaN

NaN === NaN

둘 다 NaN 이라도 **false** 가 된다.

부등 연산자

//부정

!=

!==

부등 연산자

```
var a = 1;  
var b = 2;  
a == b
```

부등 연산자

```
var a = 1;
```

```
var b = 2;
```

```
a == b
```

```
a != b
```

부등 연산자

```
var a = 1;  
var b = 1;  
a != b
```

부등 연산자

```
var a = "a";
var b = "b";
a != b
```

부등 연산자

```
var a = "a";
var b = "a";
a != b
```

Object

Object

```
var object = {};
```

Object

```
var object = { key : value };
```

Object

```
var person = {  
    name : "김용원",  
    job : "교수",  
    phone : "010-9137-8688",  
    email : "rh@102labs.com"  
};
```

Object

```
var person = {  
    "name" : "김용원",  
    "job" : "교수",  
    "phone" : "010-9137-8688",  
    "email" : "rh@102labs.com"  
};
```

```
person.name;  
person.job;  
person.phone;  
person.email;
```

```
person["name"];
person["job"];
person["phone"];
person["email"];
```

Array

Array 객체

한 번에 두가지 이상의 값을 포함할 수 있는 객체

사용빈도가 아주 높다.

Array 배열

```
var a = 10;  
var b = 'apple';  
var c = null;  
var d = a;  
var _array = [a, b, c, d];  
console.log(_array[3]);
```

if

if 조건문

if 안의 조건이 **true** 인 경우.

if 안의 조건이 **false** 인 경우.

true 일 경우 if 조건 안의 코드가 실행.

false 일 경우 else 안의 코드가 실행.

if 조건문

```
var a = 3;  
var b = 10;  
var c;  
if (typeof c === 'undefined') {  
    c = a % b;  
};
```

if 조건문

```
var a = 3;  
var b = 10;  
var c = 0;  
if (typeof c === 'undefined') {  
    c = a % b;  
};
```


for

동일한 동작을 반복 수행할 수 있도록 도와주는 구문

for

while

for 반복문

```
var _array = ['a', 'b', 'c', 'd', 'e'];
for (var i = 0; i < _; i++) {
    console.log(_);
};
```

for 반복문

```
var _array = ['a', 'b', 'c', 'd', 'e'];
for (var i = 0; i < 5; i++) {
    console.log(_array[i]);
};
```

for 반복문

```
var _say = ['Hello', 'My', 'Name', 'is', '__'];
```

Hello My Name is ____.

위와 같이 출력되도록 반복문을 이용해서 작성해봅시다.

for 반복문

```
var _say = ['Hello', 'My', 'Name', 'is', '__'];
var say = '';
for (var i = 0; i < 5; i++) {
    say += _say[i];
}
console.log(say);
```

for 반복문

```
var _say = ['Hello', 'My', 'Name', 'is', '__'];
var say = '';
for (var i = 0; i < 5; i++) {
  if (i !== 0) {
    say += ' ';
  };
  say += _say[i];
  if (i === 4) {
    say += '.';
  };
};
console.log(say);
```

function

Function 함수

재사용을 위해 코드 블록을 감싸는 방법.

작업을 수행하거나 값을 계산하는 등의 역할을 수행하는 코드를 포함.

호출을 통해 내부의 코드를 동작시킨다.

유효범위를 가지고 있으며, 실행 시 값을 반환하도록 되어 있다.

function 함수

```
function name() {  
};  
name();
```

function 함수

```
function name(param) {  
};  
name();
```

function 함수

```
var a = 20, b = 30, c = 10;
console.log(c);
function addNumbers() {
    c = b + a;
}
console.log(c);
addNumbers();
console.log(c);
```

scope

Scope 유효범위

코드의 참조 범위.

변수와 매개변수의 접근성과 생존 기간을 의미.

전역변수(범위) / 지역변수(범위)로 구분.

전역범위는 스크립트 내의 모든 곳이 참조 가능.

지역범위는 지정된 함수 내부에서만 참조 가능.

```
var global = 'global';
var a = 'My';
var b = 'Name';
function local() {
    var local = 'local';
    var a = 'Hello';
    var b = 'World';
    console.log(a + ' ' + b);
}
local();
console.log(a + ' ' + b);
```

```
var global = 'global';
var a = 'My';
var b = 'Name';
function local() {
    var local = 'local';
    var a = 'Hello';
    var b = 'World';
    console.log(a + ' ' + b);
};
local();
console.log(a + ' ' + b);
```

Global

```
var global = 'global';
var a = 'My';
var b = 'Name';
function local() {
    var local = 'local';
    var a = 'Hello';
    var b = 'World';
    console.log(a + ' ' + b);
};

local();
console.log(a + ' ' + b);
```

Local

Global

```
var global = 'global';
function local() {
    var local = 'local';
};
```

method

Method 메서드

작업을 수행하거나 값을 연산하는 등의 역할을
수행하는 코드를 포함하고 있는 블럭단위의 뭉치.

함수와 의미적으로 동일하나, 객체에 의존되어 있는 함수를 칭함.

```
function sayHello() {  
    console.log('Hello~');  
};  
sayHello();
```

Function

method 메서드

```
var _obj = {  
    sayHello : function() {  
        console.log('Hello!');  
    }  
};  
  
_obj.sayHello();
```

Method

함수의 종류

JavaScript에서 함수는 총 3가지 종류.

1. 명시적 함수
2. 익명 함수 (함수 생성자)
3. 함수 리터럴 (함수식)

명시적 함수

```
function sayHello(name) {  
    return 'Hello ' + name;  
};  
var say1 = sayHello('World');  
var say2 = sayHello('Everyone');  
console.log(say1);  
console.log(say2);
```

익명 함수 (함수 생성자)

```
var sayHello = new Function('name', 'return "Hello " + name');  
sayHello('World');
```

함수 리터럴 (함수식, 익명함수)

```
var sayHello = function(name) {  
    return 'Hello ' + name;  
};
```

함수 작성법

선언식 / 표현식

일반적 함수는 선언식.

메서드 함수는 표현식.

```
//함수 :: 선언식.  
function sayHello() {  
    console.log('Hello~');  
};
```

```
//함수 :: 표현식  
var sayHello = function() {  
    console.log('Hello!');  
};
```

함수 호출(실행)

함수, 메서드 모두 동일하게 () 를 이용하여 호출(실행)함으로써
내부의 코드를 동작시킴.

```
//함수 :: 선언식.  
function sayHello() {  
    console.log('Hello~');  
};  
//함수 호출.  
sayHello();
```

```
//함수 :: 표현식  
var sayHello = function() ·  
    console.log('Hello!');  
};  
//함수 호출.  
sayHello();
```

```
//메서드  
var _obj = {  
    sayHello : function(){  
        console.log('Hello?');  
    }  
};  
//메서드 함수 호출.  
_obj.sayHello();
```

매개변수

Parameter 매개변수

변수의 한 종류로, 함수에 전달되는 여러 데이터 중 하나를 의미함.

매개변수의 목록은 함수를 정의하는 부분에 포함되며, 매 함수 호출시 함수에 주입된다.

```
function sendMessage(msg) {  
    console.log(msg);  
};
```

전달인자

Argument 전달인자

함수에 정의된 매개변수를 통해 전달되는 실제 값.

```
function sendMessage(msg) {  
    console.log(msg);  
};  
sendMessage('Hello');
```

```
function sendMessage('Hello') {  
    console.log(msg);  
};  
sendMessage('Hello');
```

```
function sendMessage( 'Hello' ) {  
    console.log( 'Hello' );  
};  
sendMessage( 'Hello' );
```

매개변수 유효성

매개변수 유효성

함수에 전달 시 사용되는 매개변수는 실행시점에 유효할 수도 유효하지 않을 수도 있다.
코드 실행 시 유효성을 검증할 필요가 있다.

```
function sendMessage(msg) {  
    console.log(msg);  
};  
sendMessage('Hello');
```

```
function sendMessage(msg) {  
    console.log(msg);  
};  
sendMessage();
```

```
function sendMessage(msg) {  
    if (typeof msg === 'undefined') {  
        console.log('No Message');  
    } else {  
        console.log(msg);  
    };  
};  
sendMessage();
```

매개변수 활용

매개변수는 어떤 값도 전달인자로 대입될 수 있으며,
전달할 수 있는 매개변수는 다수로 사용이 가능하다.

정의된 매개변수의 수보다 많은 수의 값을 전달할 경우도 문제없이 실행 가능.
정의된 수보다 전달인자가 적을 경우 값이 없는 매개변수는 **undefined** 가 할당된다.

```
function sum(number1, number2) {  
    console.log(number1 + number2);  
};  
sum(1, 2);
```

```
function sum(number1, number2) {  
    console.log(number1 + number2);  
};  
sum(1, 2, 3, 4, 5, 6);
```

```
function sum(number1, number2) {  
    console.log(number1 + number2);  
};  
sum(1);
```

함수 반환

함수는 실행시 반드시 결과를 반환한다.

return 문을 이용하여 반환할 값을 반환하며,

return 문이 없을 경우, undefined 를 반환한다.

```
function sumPrices(price1, price2) {  
    price1 + price2;  
};  
var price = sumPrices(10, 20);  
console.log(price);
```

```
function sumPrices(price1, price2) {  
    return price1 + price2;  
};  
var price = sumPrices(10, 20);  
console.log(price);
```

```
function getSize(  ) {  
    ;  
};  
var size = getSize(30, 32);  
console.log(size);
```


논리 연산자

```
var a = 0;  
var b = 1;  
var c = 2;  
var d = 2;  
  
// && - and  
if(a !== b && c === d){  
    console.log('두 조건 중 모두 일치합니다.');//  
}
```

```
var a = 0;  
var b = 1;  
var c = 2;  
var d = 2;  
  
// || - or  
if(a === b || c === d){  
    console.log('두 조건 중 하나는 일치합니다.');//  
}
```

Switch

Switch 조건문

```
var greeting = 0;
switch (greeting){
    case 0 :
        console.log('Hello');
        break;
    case 1 :
        console.log('World');
        break;
    case 2 :
        console.log(':^)');
        break;
    default :
        console.log('X(');
        break;
}
```

While

While 반복문

```
var i = 0;  
while(i < 10){  
    console.log(i);  
    i++  
}
```

Array

Array 객체

한 번에 두가지 이상의 값을 포함할 수 있는 객체

사용빈도가 아주 높다.

Array 배열

```
var a = 10;  
var b = 'apple';  
var c = null;  
var d = a;  
var _array = [a, b, c, d];  
console.log(_array[3]);
```

```
function getMembers(){
    return ['rh', 'june', 'mind'];
}

var members = getMembers();
console.log(members[0]);
console.log(members[1]);
console.log(members[2]);
```

배열의 추가 / 제거 / 정렬

unshift(); push(); shift(); pop();
concat(); splice(); sort(); reverse();

```
var _heros = ['Iron Man', 'Hulk', 'Thor', 'Doctor Strange'];
_heros.unshift('Captain America');
_heros.push('Spider-Man');
_heros.concat(['Black Panther', 'Ant-Man']);
_heros.splice(2, 0, 'Vision');
_heros.splice(2, 1, 'Loki');
_heros.shift();
_heros.pop();
```

```
_array.splice(start, deleteCount, string[]);
```

```
var _heros = ['Iron Man', 'Hulk', 'Thor', 'Doctor Strange'];
_heros.sort();
_heros.reverse();
```

배열, 객체의 반복문

for ... in

```
function getMembers(){
    return ['rh', 'june', 'mind'];
}

var members = getMembers();
for(var i = 0; i < members.length; i++){
    console.log(members[i]);
}
```

```
var _heros = ['Iron Man', 'Hulk', 'Thor', 'Doctor Strange'];
for(var name in _heros){
    console.log(name);
}
```

```
var _person = {
    name : '김용원',
    job : '교수',
    phone : '010-9137-8688',
    email : 'rh@102labs.com'
};

for(var key in _person){
    console.log(key + ' : ' + _person[key]);
}
```

Math

Math

Math.pow	제곱
Math.round	반올림
Math.ceil	올림
Math.floor	내림
Math.sqrt	제곱근
Math.random	랜덤

```
Math.pow(3, 2);
Math.round(1.4);
Math.ceil(1.2);
Math.floor(1.2);
Math.sqrt(9);
Math.random();
```

Math.pow(3, 2);	3 의 2 제곱
Math.round(1.4);	1.4 의 반올림
Math.ceil(1.2);	1.2 의 올림
Math.floor(1.2);	1.2 의 내림
Math.sqrt(9);	9 의 제곱근
Math.random();	0~1.0 사이의 랜덤한 숫자

```
Math.round(100 * Math.random());
```

```
Math.abs(-100 + 50);
```

```
Math.max(0, 10);
```

```
Math.min(0, -10);
```

```
Math.max.apply(null, [0, 2, 3, 4, 5]);
```

```
Math.min.apply(null, [-5, -4, -3, -2, -1, 0]);
```

Apply

Apply

```
var fruit = { apple : 1000, orange : 2000, lemon : 3000, mango : 4000};  
function sum(){  
    var result = 0;  
    for(var key in this){  
        result += this[key];  
    }  
    return result;  
}  
console.log(sum.apply(fruit));
```

