



# INTERACTION PROGRAMMING 1

인터랙션 프로그래밍 1

---

*6 Week.*

*2020. 5. 7.*

# JavaScript Review

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.



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

```
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();
```

```
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

scope 유효범위

---

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

**method**

## Method 메서드

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

함수와 의미적으로 동일하나, 객체에 의존되어 있는 함수를 칭함.

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

Function



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

Method

# 함수의 종류

JavaScript 에서 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 호출(실행)

---

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

## 함수 호출(실행)

---

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

## 함수 호출(실행)

---

//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 변환

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.



```
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();`

## Array 배열의 제어

---

```
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[]);
```

## Array 배열의 제어

---

```
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]);  
}
```



## Array 배열의 반복

---

```
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            제공

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();
```

<code>Math.pow(3, 2);</code>	3 의 2 제곱
<code>Math.round(1.4);</code>	1.4 의 반올림
<code>Math.ceil(1.2);</code>	1.2 의 올림
<code>Math.floor(1.2);</code>	1.2 의 내림
<code>Math.sqrt(9);</code>	9 의 제곱근
<code>Math.random();</code>	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));
```



**BOM**

# BOM

(Browser Object Model)

window

navigator

screen

history

location

**window**

window

---

# window

**BOM(DOM Level 0) 의 최상위 객체**

브라우저 객체의 최상위 객체

브라우저 환경의 모든 객체 속성을 담고 있다.

# document

## BOM(DOM Level 0) 의 최상위 객체

브라우저 객체의 최상위 객체

브라우저 환경의 모든 객체 속성을 담고 있다.

브라우저 문서(html)를 모두 읽어들이면 해당 문서를 객체화하여 **document** 객체가 된다.

html 문서의 root 객체로 모든 node 를 갖게 된다.



window

---

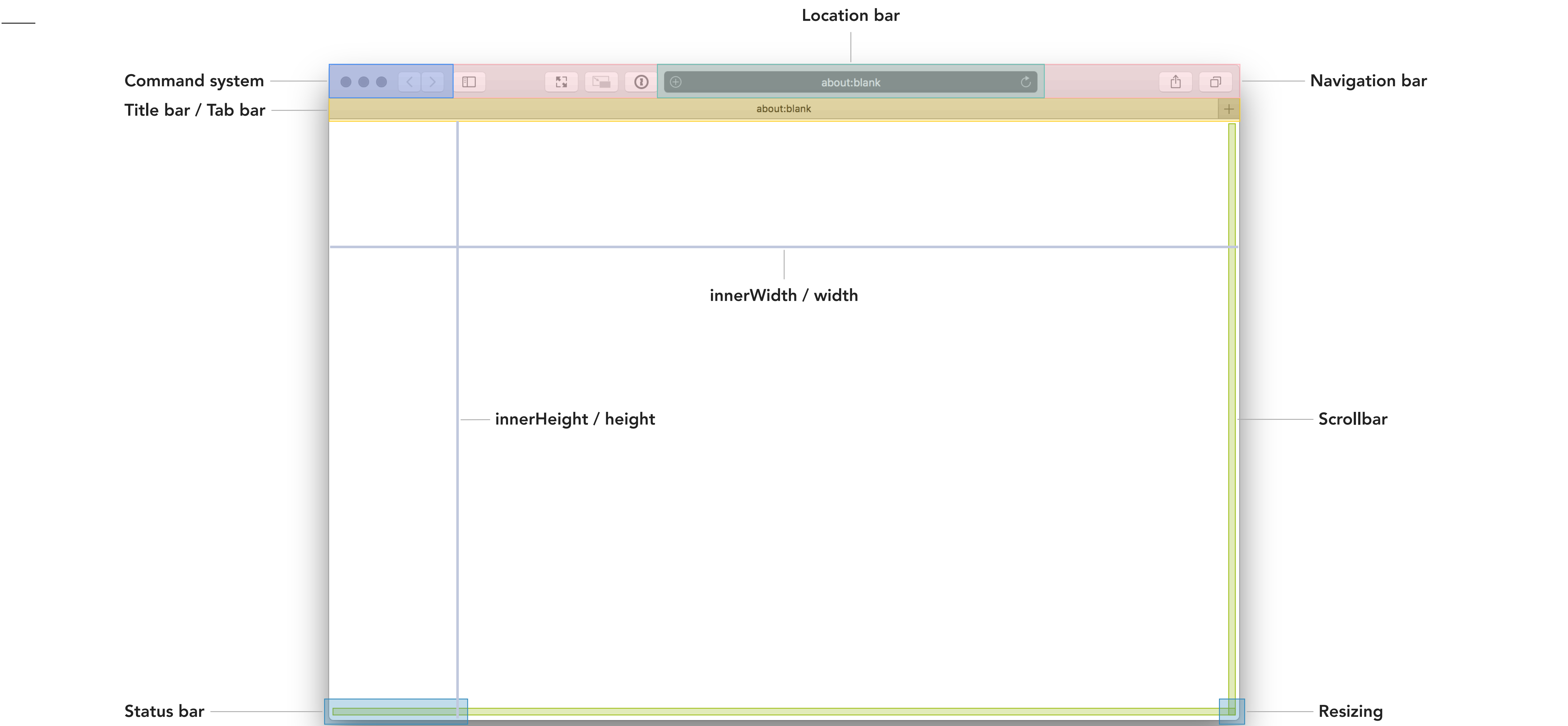
## window.innerWidth

윈도우 콘텐츠 영역의 너비 값(pixel)을 반환한다.

## window.innerHeight

윈도우 콘텐츠 영역의 높이 값(pixel)을 반환한다.

window



## window.screenTop

모니터상에서 브라우저의 종(수직) 좌표 값(pixel)을 반환한다.

## window.screenLeft

모니터상에서 브라우저의 횡(수평) 좌표 값(pixel)을 반환한다.

## window.pageXOffset

문서 내부의 횡(수평) 스크롤 좌표 값(pixel)을 반환한다.

## window.pageYOffset

문서 내부의 종(수직) 좌표 값(pixel)을 반환한다.

window

---

**window.location**

현재 웹페이지의 URL 객체를 반환한다.

window

---

## window.alert();

```
window.alert('message');
```

OK 버튼을 포함하고 있는 경고창을 출력한다.

## window.prompt();

```
window.prompt('message');
```

메세지와 사용자가 입력 가능한 프롬프트를 포함한 다이얼로그를 출력한다.

## window.confirm();

window.confirm('message');

메세지와 '확인', '취소' 버튼을 포함하고 있는 다이얼로그를 출력한다.

## window.scrollTo(x, y);

window.scrollTo(0, 0);

문서를 입력한 위치(x, y) 좌표로 스크롤한다.

## window.setInterval();

window.setInterval(callback, time);

주기적으로 실행되는 타이머를 생성한다. (ms - milliseconds 단위)

## window.clearInterval();

window.clearInterval(timer);

setInterval 로 생성된 타이머를 초기화한다.



## window.setTimeout();

window.setTimeout(callback, time);

한번만 실행되는 타이머를 생성한다. (ms - milliseconds 단위)

## window.clearTimeout();

window.clearInterval(timer);

setTimeout 으로 생성된 타이머를 초기화한다.

## window.open();

window.open(URL, name, specs, replace);

새로운 브라우저 window 를 연다.

**URL**        새 window 의 URL

**name**        window 의 이름 (\_blank, \_parent, \_self, \_top, name)

**specs**        선택적인 매개변수

**replace**     현재 방문이력의 목록에서 대체 여부

window

**window.open();**

specs

top=pixels left=pixels width=pixels height=pixels

titlebar=yes|no|1|0 menubar=yes|no|1|0 scrollbars=yes|no|1|0 toolbar=yes|no|1|0

location=yes|no|1|0 status=yes|no|1|0 resizable=yes|no|1|0 fullscreen=yes|no|1|0

channelmode=yes|no|1|0 directories=yes|no|1|0

window

---

**window.close();**

window.close();

target.close();

새로 열린 window 를 닫는다.

**navigator**

# window.navigator

브라우저의 정보를 담고 있는 객체

window.navigator.appName

브라우저의 이름을 반환한다.

window.navigator.appVersion

브라우저의 버전 정보를 반환한다.

# window.navigator.userAgent

window.navigator.userAgent

target.navigator.userAgent

브라우저의 기본정보를 담고 있는 navigator 의 프로퍼티

해당 내용을 바탕으로 현재 문서가 열려있는 브라우저의 종류 및 버전, 사용중인 OS 정보까지 확인이 가능하다.



**screen**

screen

---

# window.screen

브라우저 화면의 정보를 담고 있는 객체

**window.screen.availWidth**

화면의 너비를 반환한다. (작업표시줄 등을 제외한 높이)

**window.screen.availHeight**

화면의 높이를 반환한다. (작업표시줄 등을 제외한 높이)

screen

---

`window.screen.width`

화면의 너비를 반환한다.

`window.screen.height`

화면의 높이를 반환한다.

**history**

# window.history

사용자가 방문한 URL 등의 이력 정보를 담고 있는 객체

history

---

window.history.length

방문이력의 수를 반환한다.

```
window.history.back();
```

이전 방문이력을 로드한다.

```
window.history.forward();
```

다음 방문이력을 로드한다.



`window.history.go();`

`window.history.go(index);`

방문이력 중 `index` 에 위치한 이력의 URL 로 이동한다.

**location**

location

---

# window.location

현재 URL 에 대한 정보를 담고 있는 객체

location

---

`window.location.hash`

URL 에 포함된 hash 값을 반환한다. (#)

`window.location.host`

URL 의 호스트네임과 포트를 반환한다. (80 포트는 생략됨)

location

---

`window.location.hash`

URL 에 포함된 hash 값을 반환한다. (#)

`http://host.com/path?key=value#hash`

location

---

`window.location.host`

URL 의 호스트네임과 포트를 반환한다. (80 포트는 생략됨)

`http://host.com:port/path?key=value#hash`

**window.location.hostname**

URL 의 호스트네임을 반환한다.

**window.location.href**

URL 전체를 반환합니다. 값을 할당하는 경우 해당 경로로 이동

location

---

`window.location.hostname`

URL 의 호스트네임을 반환한다.

`http://host.com/path?key=value#hash`



## window.location.href

URL 전체를 반환합니다. 값을 할당하는 경우 해당 경로로 이동

`http://host.com/path?key=value#hash`

`window.location.href = http://host.com;`

location

---

`window.location.origin`

프로토콜을 포함한 호스트네임과 포트를 반환한다.

`window.location.pathname`

URL 중 path 를 반환합니다.

location

---

`window.location.origin`

프로토콜을 포함한 호스트네임과 포트를 반환한다.

`http://host.com/path?key=value#hash`

location

---

`window.location.pathname`

URL 중 path 를 반환합니다.

`http://host.com/path?key=value#hash`

location

---

window.location.port

URL 중 포트를 반환합니다.

window.location.protocol

URL 의 프로토콜을 반환합니다.

location

---

window.location.port

URL 중 포트를 반환합니다.

http://host.com:port/path?key=value#hash

# window.location.protocol

URL 의 프로토콜을 반환합니다.

http://host.com/path?key=value#hash

https://host.com/path?key=value#hash

location

---

window.location.search

URL 의 Query 문자열을 반환한다.



location

---

`window.location`.search

URL 의 Query 문자열을 반환한다.

`http://host.com/path?key1=value&key2=value#hash`

`window.location.assign();`

`window.location.assign(url);`

새로운 문서를 로드한다. (방문이력유지)

`window.location.reload();`

현재 문서를 새로 고침한다.

location

---

`window.location.replace();`

`window.location.replace(url);`

새로운 문서를 로드한다. (방문이력대체)



**DOM**

# DOM (Document Object Model)

DOM 은 문서의 구조화된 표현을 제공하며 프로그래밍 언어가  
DOM 구조에 접근할 수 있는 방법을 제공하여 문서 구조, 스타일, 내용 등을 변경할 수 있게 돕는다.

DOM 은 구조화된 nodes 와 property 와 method 를 갖고 있는 objects 로 문서를 표현한다.

웹 페이지를 스크립트 또는 프로그래밍 언어로 사용될 수 있게 연결시켜 주는 역할을 담당한다.

**DOM === window.document**

## document.documentElement

document 객체 자신의 element 노드를 반환한다.

## document.body

document 객체의 하위 객체 중 body element 노드를 반환한다.



**document.cookie**

document 객체의 cookie 정보를 반환한다.

**document.domain**

document 객체의 domain 정보를 반환한다.

## document.scripts

document 객체의 하위 객체 중 script element 노드 목록을 반환한다.

## document.title

document 객체 title 노드의 값을 반환한다.

**document.createElement();**

`document.createElement('div');`

element 노드를 새로 생성한다.

**document.getElementById();**

`document.getElementById('id');`

document 하위의 element 노드 객체 중 매개변수의 전달인자와 같은 이름의 id 를 가진 첫 번째 element 객체 1개를 반환한다.

# `document.getElementsByClassName();`

`document.getElementsByClassName('class');`

`document` 하위의 `element` 노드 객체 중 매개변수의 전달인자와 같은 이름의 `class` 를 가진 `element` 노드 목록을 반환한다.

# `document.getElementsByName();`

`document.getElementsByName('name');`

`document` 하위의 `element` 노드 객체 중 매개변수의 전달인자와 같은 이름의 `name` 속성을 가진 `element` 노드 목록을 반환한다.

# `document.getElementsByTagName();`

```
document.getElementsByTagName('input');
```

document 하위의 element 노드 객체 중 매개변수의 전달인자와 같은 이름의 tag 를 가진 element 노드 목록을 반환한다.

# `document.write();`

```
document.write('hello');
```

document 하위 객체 중 body 에 전달된 String 문자열을 출력한다.

# DOM Element

문서상의 HTML Element 를 의미한다.

## element.attributes

element 노드의 속성들을 담고 있는 NamedNodeMap 객체를 반환한다.  
(NamedNodeMap 은 속성 객체의 컬렉션 객체)

## element.childNodes

element 객체의 자식 노드들을 담고 있는 배열을 반환한다.

## `element.children`

`element` 객체의 자식 노드들 중 `element` 노드들만을 포함하고 있는 배열을 반환한다.

## `element.classList`

`element` 객체의 `class` 속성의 값을 담고 있는 `DOMTokenList` 객체를 반환한다.



**element.classList.add();**

**element.classList.add(class1, class2, ...);**

**element** 객체의 **classList** 객체에 하나 이상의 **class** 를 추가하는 메서드.

**element.classList.remove();**

**element.classList.remove(class1, class2, ...);**

**element** 객체의 **classList** 객체에 하나 이상의 **class** 를 삭제하는 메서드.

```
element.classList.toggle();
```

```
element.classList.toggle(class1, true|false);
```

element 객체의 classList 객체에 class 를 toggle 로 add/remove 할 수 있도록 하는 메서드.

```
element.classList.contains();
```

```
element.classList.contains('class');
```

element 객체의 classList 객체 중 매개변수의 전달인자와 같은 이름의 Class 를 가진 요소의 포함 여부를 Boolean 으로 반환한다.

## element.id

element 객체의 id 속성을 문자열로 반환한다.

## element.className

element 객체의 class 속성을 문자열로 반환한다.

## `element.firstChild`

`element` 객체의 첫번째 자식 노드를 반환한다.

## `element.firstChildElement`

`element` 객체의 첫번째 `element` 객체를 반환한다.

## `element.lastChild`

`element` 객체의 마지막 자식 노드를 반환한다.

## `element.lastElementChild`

`element` 객체의 마지막 `element` 객체를 반환한다.

## element.innerHTML

element 객체 내부의 HTML 을 문자열로 반환한다.

## element.style

element 객체의 style 속성의 값을 반환한다.

**element.addEventListener();**

`element.addEventListener('event', callback);`

element 노드에 이벤트 핸들러를 할당한다.

**element.appendChild();**

`element.appendChild(element1);`

element 노드 객체에 element1 노드를 추가한다.

**element.click();**

element 노드에 click 이벤트를 발생시킨다.

**element.cloneNode();**

element 노드를 복사한 element 를 반환한다.



**element.contains();**

**element.contains(element1);**

**element** 노드에 **element1** 노드의 포함 여부를 **Boolean** 으로 반환한다.

**element.getAttribute();**

**element.getAttribute('attribute1');**

**element** 노드 **attribute1** 속성 값을 반환한다.

**element.getAttributeNode();**

`element.getAttributeNode('attribute1');`

element 노드의 속성 노드들 중 attribute1 노드 객체를 반환한다.

**element.getElementsByClassName();**

`element.getElementsByClassName('class');`

element 노드의 자식 element 중 class 를 포함하고 있는 element 노드 목록을 반환한다.

**element.hasAttributes();**

element 노드가 속성 노드를 포함하고 있는지 여부를 Boolean 으로 반환한다.

**element.insertBefore();**

**element.insertBefore(element1, element2);**

element 노드 내부에 element1 을 element2 의 앞쪽 위치에 추가한다.

**element.querySelector();**

`element.querySelector('query');`

element 노드의 자식 노드들 중 query 에 해당하는 노드를 탐색하여 그 첫번째 노드를 반환한다.

**element.querySelectorAll();**

`element.querySelectorAll('query');`

element 노드의 자식 노드들 중 query 에 해당하는 노드를 탐색하여 노드 목록을 반환한다.

**element.removeAttribute();**

**element.removeAttribute('attribute1');**

**element** 노드의 attribute1 속성을 삭제한다.

**element.removeChild();**

**element.removeChild(node);**

**element** 노드의 자식 노드들 중 **node** 를 삭제한다.

**element.removeEventListener();**

`element.removeEventListener('event', callback);`

element 노드의 event 이벤트에 할당된 이벤트 핸들러를 삭제한다.

**element.setAttribute();**

`element.setAttribute('attribute1', 'value');`

element 노드에 attribute1 속성을 추가하고 그 값은 value 로 할당한다.

**element.toString();**

element 노드를 String 으로 변환한다.

