

JavaScript 기초

- 강사 김현오 -



- 이름 : 김현오 (hyunohkim@nextree.co.kr)
- 회사 : 넥스트리소프트 선임엔지니어
- **경력 : 7년차 소프트웨어 엔지니어**
- 분야 : Java/웹 기반 개발, 객체/컴포넌트 모델링
- 회사

✓ 넥스트리소프트 (현)
<http://nextree.co.kr>



• 최근 주요 프로젝트

- ✓ 2013 – 삼성화재 경영계획관리 시스템
- ✓ 2012 – 삼성화재 프로젝트 지원센터 구축
- ✓ 2011 – 삼성화재 백오피스 모바일화
- ✓ 2010 – 플라워팀 리본 출력모듈 개발
- ✓ 2008 – 푸르덴셜 생명보험 차세대

• 외부활동:

- ✓ 강의 : KOSTA 야간 강의 (재직자 대상)
 - UML과 시스템 구축
 - Design Pattern
 - 데이터접근프레임워크
 - JavaScript 패턴
 - Java 프로그래밍

목차

1. JavaScript 소개
2. JavaScript 어휘 구조
3. 타입, 값, 변수
4. 표현식과 연산자
5. 구문
6. 함수
7. JavaScript 내장 객체
8. Window 객체
9. DOM



1. JavaScript 소개

1.1 JavaScript 소개

1.2 JavaScript 개요

- JavaScript 소개

JavaScript 소개

✓ JavaScript는 Scripting Language 입니다.

- 가볍고 손쉽게 작성이 가능한 프로그램 언어입니다.
- HTML 내에 프로그래밍 코드를 삽입하여 사용
- 사실상 현존하는 거의 모든 브라우저 에서 이를 실행
- 스크립트는 쉽게 배워 사용 가능 (어렵게 사용하기도 합니다...)

✓ JavaScript로 무엇을 할 수 있을까요?

- HTML 화면을 만들 수 있습니다.
- 사용자의 이벤트에 반응하게 해줍니다.
- HTML 내용을 변경
- 이미지를 변경하기도 하구요.
- HTML의 Style을 바꾸기도
- 그리고 잘못 입력된 내용을 확인





1. JavaScript 소개

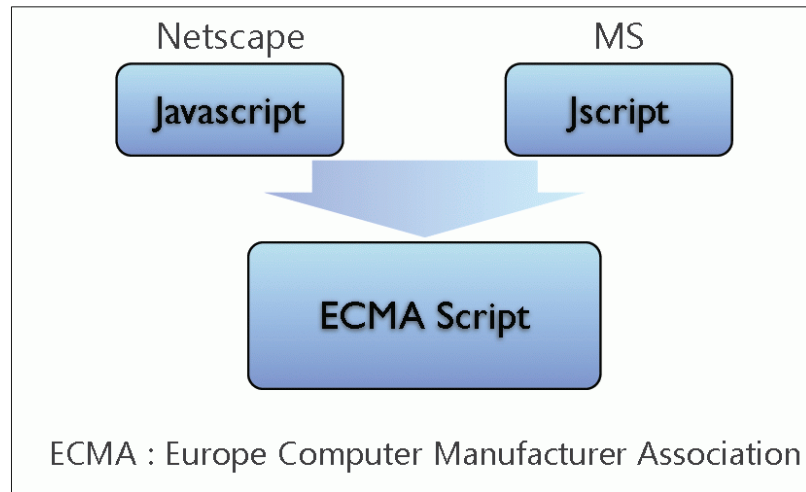
1.1 JavaScript 소개

1.2 JavaScript 개요

- JavaScript 개요
- HTML과 JavaScript
- ECMAScript
- JavaScript의 미래

JavaScript 개요

- ✓ 프로토타입 기반의 객체지향 스크립트 프로그래밍 언어
- ✓ 웹 브라우저 지원 : JavaScript는 HTML과 함께 클라이언트에 다운로드되어 실행됨
- ✓ HTML에서 사용하기 위한 예약어
 - HTML의 헤드부분에서 <script> 태그사용
 - HTML안에 JavaScript 코드를 포함하거나 외부의 JavaScript 파일을 참조하여 사용
 - Embedding : <script type="text/javascript"> ... code ... </script>
 - Referring : <script type="text/javascript" src="url"></script>
- ✓ JavaScript, ECMAScript, Jscript 관계
 - ECMAScript
 - ECMA(European Computer Manufacturers Association)의 ECMA-262 스펙에 대한 자바스크립트 표준
 - JavaScript나 JScript는 ECMAScript에 부합되도록 함
 - JavaScript
 - ECMAScript에 기반하여 구현된 Netscape사의 스크립트 언어
 - Jscript
 - Microsoft사의 스크립트 언어
 - Internet Explorer에서 사용하는 JavaScript는 결국 JScript로 구동됨



JavaScript 개요

✓ 동작원리

- 자바스크립트가 삽입된 문서를 브라우저가 읽어 들일 때 해석함
- 이 때 즉시 실행될 수도 있고 어떤 이벤트에 의해 실행될 수도 있음

✓ 장점

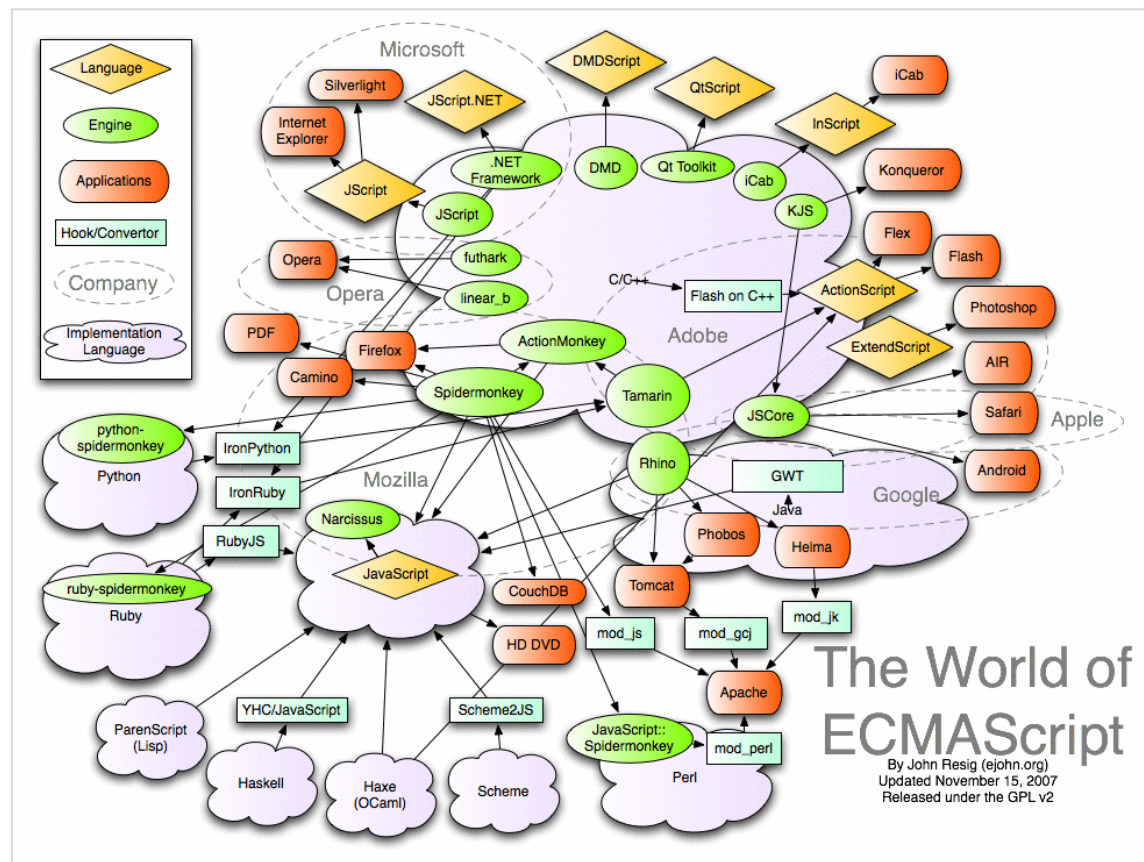
- 다른 언어에 비해 간단하고 배우기 쉬움
- 컴파일 과정이 없으므로 신속한 개발가능
- HTML과 같이 플랫폼에 독립적
- 시스템의 부담이 적음

✓ 단점

- 코드의 보안성
- 내장함수의 한계
- 디버깅 및 개발도구의 부족

✓ 특징

- 이벤트 중심 프로그래밍
- 객체 기반(지향) 언어



[ECMAScript 관련 기술 지도 via John Resig]

HTML 문서와 JavaScript

✓ HTML 문서 내에서 JavaScript의 위치

- Script 태그
 - <script>태그는 JavaScript를 HTML에 연결하는 방법임
 - src 속성 : 독립된 자바스크립트 파일위치 지정
 - type 속성 : text/javascript (자바스크립트로 작성된 평범한 텍스트임을 명시함)

```
<script src="common.js" type="text/javascript">
```

HTML 문서와 JavaScript

✓ HTML 문서 내에서 JavaScript의 위치

- Script 태그의 위치
 - HTML 내의 어느 부분에 삽입해도 가능하나 <head> 태그와 <body> 태그 사이에 삽입하는 것을 권장
 - 헤더 부분에 위치한 자바스크립트 프로그램은 브라우저의 각종 입/출력 발생 이전에 초기화되므로 브라우저에 의해 먼저 점검됨

```
<html>
  <head>
    <title>script tag의 위치</title>
    <!-- script tag의 위치 #1 -->
    <script type="text/javascript">
      document.write("반갑습니다. 여러분!");
    </script>
  </head>
  <body>
    즐거운 하루입니다.
    <!-- script tag의 위치 #2 -->
    <script type="text/javascript">
      document.write("자바스크립트 공부하기 참~ 좋은 날씨입니다.");
    </script>
  </body>
</html>
```

<head>태그 사이에 삽입된 <script>태그

<body>태그 사이에 삽입된 <script>태그

ECMAScript

✓ 표준화 진행 상황

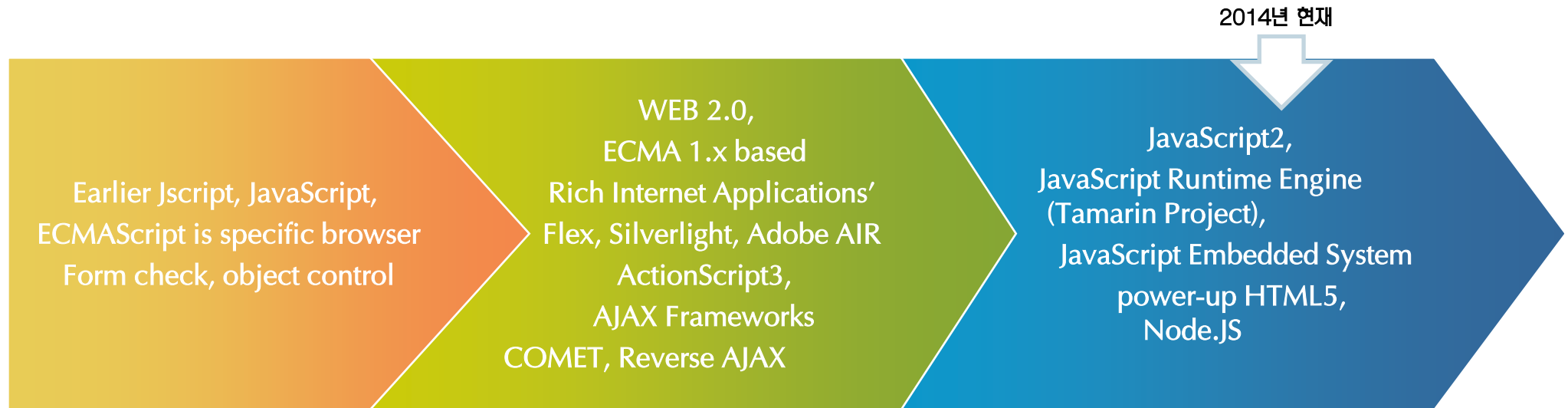
- 1997년 ECMA가 JavaScript를 표준화
- 1998년 ISO 표준으로 개정 (Edition 2)
- 1999년 추가 사항 (Edition 3)
- 2004년 6월 E4X (ECMAScript for XML)
 - first class object subtype and literal syntax
- 2006년 ECMA Edition 4 제정 (언어에 얹힌 정치적 차이로 버려짐)
- 2009년 "strict mode"를 추가(Edition 5)
- 2011년 Edition 5.1(현재까지 최신버전)
- ECMAScript 표준의 하모니(Harmony)버전은 현재 작업 중

✓ 주요 변화 상황

- 대부분 렌더링 엔진은 ECMAScript 를 99% 이상 지원
 - 2012년부터 최신 브라우저들은 모두 ECMAScript 5.1의 기능을 전부 지원
 - 예전브라우저는 최소한 ECMAScript 3까지는 지원
- Adobe가 ActionScript Engine을 오픈 소스로 제공
- 대부분 JS 엔진은 오픈 소스로 나오고 있음

JavaScript의 미래

✓ RIA 세상의 공용어 : JavaScript





2. JavaScript 어휘구조

2.1 문자 집합

2.2 주석

2.3 리터럴

2.4 식별자와 예약어

2.5 선택적인 세미콜론 사용

- 문자 집합

문자 집합

✓ 유니코드 사용

- ECMAScript3 표준 : Unicode2.1 이상 지원해야 함
- ECMAScript5 표준 : Unicode3 이상 지원해야 함
 - unicode : 지구상 사용되는 대부분의 문자를 표현할 수 있음

✓ 대소문자 구분

- 자바 스크립트는 대소문자를 구분하는 언어이다
- HTML은 대소문자 구별하지 않음 (단, XHTML은 구분)
 - 자바스크립트와 밀접한 연관이 있으므로 HTML 에서도 대소문자 구별하듯이 사용하자

✓ 공백, 개행, 제어 문자

- 토큰들 사이의 공백들을 무시 (타 언어들과 같음, 가독성 증가)

✓ 유니코드 이스케이프 시퀀스

- 유니코드를 사용 할 수 있게 해 줌
- \u 로 시작 16진수 4자리 사용 (ex \u00E9)



2. JavaScript 어휘구조

2.1 문자 집합

2.2 주석

2.3 리터럴

2.4 식별자와 예약어

2.5 선택적인 세미콜론 사용

- 주석

주석

✓ 주석 (Comment)

- 프로그램의 중간에 설명문 또는 기타 내용들을 삽입함
- 브라우저는 이를 실행코드로 인식하지 않고 무시함
- 한 줄의 주석은 (//)뒤에 표기함
- 여러 줄의 주석은 (/*) 와 (*/) 사이에 표기함 (C, Java 와 동일함)

```
<html>
  <head>
    <title>주석문</title>
    <script type="text/javascript">
      // 한줄 주석처리

      /*
        여러 줄의 주석처리
        여러 줄의 주석처리
        C, Java와 동일함.
      */
    </script>
  </head>
  <body>
  </body>
</html>
```




2. JavaScript 어휘구조

2.1 문자 집합

2.2 주석

2.3 리터럴

2.4 식별자와 예약어

2.5 선택적인 세미콜론 사용

- 리터럴 개요
- 정수 Literal
- 실수 Literal
- 문자열 Literal
- 함수 Literal
- 객체 Literal
- 배열 Literal
- 정규표현식 Literal

리터럴, 정수 Literal

✓ Literal

- 프로그램에 나타나는 데이터 값
- 프로그램 내부에서 정의되어 있는 그대로 정확히 해석되어야 할 값

✓ Integer Literal

- -9007199254740992(-253)에서 9007199254740992(253) 사이에 있는(이 두 수를 포함하여) 모든 정수를 표현할 수 있다.

```
<!DOCTYPE html>
<html>
  <body>
    <script>
      console.log(parseInt(9007199254740992 + 1, 10));
      console.log(parseInt(-9007199254740992 - 1, 10));
    </script>
  </body>
</html>
```

실수 Literal

✓ Float Literal

- 자바스크립트에서 실수는 정수 부분과 소수점, 소수점 이하 부분으로 표현된다.

```
<!DOCTYPE html>
<html>
  <body>
    <script>
      console.log(3.14);
      console.log(.33333333333333);
      console.log(6.02e23);          // 6.02 x 10^23
      console.log(1.4738223E-32);   // 1.4738223 x 10^-32
    </script>
  </body>
</html>
```

문자열 Literal

✓ String Literal

- 자바스크립트에서 문자열은 0개('') 또는 하나 이상의 Unicode 문자들이 작은따옴표(') 혹은, 큰따옴표("")로 둘러싸인 시퀀스다.
- 작은 따옴표, 큰따옴표의 사용은 제한이 없으나 보통 큰따옴표를 사용한다. 단 문자열 내 큰따옴표가 필요한 경우 작은 따옴표를 사용하기도 한다.

```
<!DOCTYPE html>
<html>
  <body>
    <script>
      console.log("");
      console.log('작은 따옴표로 문자열 나타내기');
      console.log("3.14");
      console.log('문자열 안에 포함된 "따옴표"는 이렇게...');
      console.log('특수문자 사용하기\n이 것은 두 번째 줄');
      console.log('\u00E9는 유니코드 이스케이프 시퀀스로 나타낸 문자입니다.');
```


함수 Literal

✓ Function Literal

- JavaScript는 함수를 정의하는 리터럴 표기법을 제공한다.
- 함수를 정의하는 다양한 방법
 - function 키워드를 이용하는 방법
 - Function 객체를 이용하는 방법(리터럴 표기법이 아님)

```
<!DOCTYPE html>
<html>
  <body>
    <script>
      var myFunc = function(n) { return n; };
      var myFunc2 = new Function("n", "return n;"); // anti pattern
      function myFunc3(n) { return n; }
    </script>
  </body>
</html>
```



객체 Literal

✓ Object Literal

- JavaScript는 객체를 생성하고 속성(property)을 지정하는 리터럴 표기법을 제공한다.
- “{” , “}” 중괄호를 이용하며 속성 값에 리터럴 표기법으로 값을 추가할 수 있다.
- JSON 표기법으로 많이 알려져 있다.

```
<!DOCTYPE html>
<html>
  <body>
    <script>
      var myComputer = {
        name : "iMac"
        ,type : "Desktop"
      };
    </script>
  </body>
</html>
```

배열 Literal

✓ Array Literal

- JavaScript는 배열 객체를 생성하는 리터럴 표기법을 제공한다.
- 대괄호("[", "]") 는 배열을 시작과 끝을 나타내며 콤마(",")로 각 값을 구분한다.
- 배열의 값은 리터럴 표기법을 사용하여 나타낼 수 있다.

```
<!DOCTYPE html>
<html>
  <body>
    <script>
      var a = [1, 2];
      var b = [function(){ return 1; }, function() { return 2; }];
      console.log(a[0]);
      console.log(b[1]());
    </script>
  </body>
</html>
```

정규표현식 Literal

✓ RegExp Literal

- 정규 표현식을 위한 리터럴 표기법을 제공한다.

```
<!DOCTYPE html>
<html>
  <body>
    <script>
      var regexp = /[a-z]/g;
      console.log(regexp.test('a'));
      console.log(regexp.test('A'));
    </script>
  </body>
</html>
```

- * 정규 표현식이란 특정한 규칙을 가진 문자열의 집합을 표현하는 데 사용하는 형식 언어이다.



2. JavaScript 어휘구조

2.1 문자 집합

2.2 주석

2.3 리터럴

2.4 식별자와 예약어

2.5 선택적인 세미콜론 사용

- 식별자

- 예약어

식별자

✓ 식별자(Identifier)는 변수 또는 함수에 이름을 붙이거나 반복문에 레이블을 붙이는 데 사용한다.

✓ 식별자 작성 규칙

- 예약어는 사용할 수 없다.
- 숫자로 시작할 수 없다.
- 특수문자는 밑줄(_)과 달러(\$) 만 허용된다.
- 공백문자를 포함할 수 없다.

//올바른 식별자

```
i  
my_variable_name  
v13  
_dummy  
$str
```

// 잘못된 식별자

```
123      // 숫자로 시작할 수 없음  
my name  // 공백문자를 포함할 수 없음  
#value   // 특수문자는 _와 $만 허용됨  
while    // 예약어는 사용할 수 없음
```

예약어 (1/2)

✓ 예약어 (식별자로 사용할 수 없는 단어들)

break	delete	function	return	typeof	default
case	do	if	switch	var	for
catch	else	in	this	void	null
continue	false	instanceof	throw	while	null
Debugger	finally	new	true	with	

✓ 엄격한 모드(Strict Mode)에서만 예약어로 사용

implement	let	private	public	yield	interface
package	protected	static			

✓ 완전한 예약어는 아니지만 식별자 사용 제한

argument	eval
----------	------

예약어 [2/2]

✓ ECMAScript5에서는 사용가능 하지만 ECMAScript3에서는 사용불가

abstract	double	goto	native	static
boolean	enum	implements	package	super
byte	export	import	private	synchronized
char	extends	int	protected	throws
class	final	interface	public	transient
const	float	long	short	volatile

✓ 전역변수와 함수를 정의해 둔 것

arguments	encodeURIComponent	Infinity	Number	RegExp
Array	encodeURIComponent	isFinite	Object	String
Boolean	Error	isNaN	parseFloat	SyntaxError
Date	eval	JSON	parseInt	TypeError
decodeURI	EvalError	Math	RangeError	undefined
decodeURIComponent	Function	NaN	ReferenceError	



2. JavaScript 어휘구조

2.1 문자 집합

2.2 주석

2.3 리터럴


2.4 식별자와 예약어

2.5 선택적인 세미콜론 사용

- 선택적인 세미콜론 사용

선택적인 세미콜론 사용

- ✓ 구문(statement)을 구분하기 위해 세미콜론(;) 을 사용한다 (타 프로그램 언어와 동일)
- ✓ 자바스크립트 자체에서 세미콜론을 적용하므로 생략할 수 있지만 명시적으로 써주는 것을 권장한다.
(일반적으로 세미콜론 없이 해석할 수 없는 경우에 자체적으로 적용해 준다)
 - return, break, continue 는 줄 바꿈을 하지 말아야 한다.
 - 줄 바꿈을 하면 자동으로 ; 가 붙는다. 원치 않는 예외 발생
 - ++, -- 는 뒤에 오는 구문의 전치 연산자로 해석이 된다.

```
<!DOCTYPE html>
<html>
  <body>
    <script>
      // 올바른 세미콜론 사용법
      function f1() {
        return 10;
      }
      // 잘못 사용한 세미콜론 
      function f2() {
        return → 세미콜론이 자동으로 적용되어(return ;) 의도하지 않은 결과 발생
          10;
      }
      document.write("f1 result : " + f1()); // return 10
      document.write("<br>");
      document.write("f2 result : " + f2()); // return undefined
    </script>
  </body>
</html>
```




3. 타입, 값, 변수

3.1 원시타입과 참조타입

3.2 원시타입

3.3 전역 객체와 레퍼객체

3.4 타입 변환

3.5 변수선언과 유효범위

- 원시타입과 참조타입

원시 타입과 참조 타입

- ✓ 원시 타입과 참조 타입
 - 원시 타입 : boolean, 숫자, 문자열, null, undefined
 - 참조 타입 : 배열, 함수 (객체)
- ✓ 원시 타입은 값 수정 불가
- ✓ 문자열을 수정하는 모든 메서드는 새로운 문자열을 반환한다.

원시타입	값으로 비교
문자열	문자열의 길이, 각 인덱스의 문자들을 비교
객체	참조로 비교, 둘의 프로퍼티 or 원소를 비교



3. 타입, 값, 변수

3.1 원시타입과 참조타입

3.2 원시타입

3.3 전역 객체와 레퍼객체

3.4 타입 변환

3.5 변수선언과 유효범위

- 숫자
- 문자열
- boolean
- null과 undefined

숫자

- ✓ JavaScript는 정수 값과 실수 값을 구분하지 않는다. (자바스크립트의 특징 중 하나)
- ✓ 모든 숫자를 실수로 표현한다. (IEEE754 표준에서 정의한 64비트 실수)

✓ 정수 리터럴

10진수 정수	숫자 시퀀스 형태로 작성된다
16진수 정수	'0x' or '0X' (0 은 숫자) 맨 앞에 붙음
8진수 정수	'0' ('0'은 숫자) 맨 앞에 붙음 (Strict Mode 에서는 금지)

✓ 부동소수점 리터럴

지수표기법 문법	[digits] [.digits] [(E e) [(+ -)] digits]
----------	---

- 예 : 110.34(소수), 1.2E+10(지수)
- 실수에 이어 E | e 뒤에 + | - 가 나오고 정수 지수 값이 나온다
- 실수에 10을 지수횟수만큼 곱한 값

숫자

✓ 산술 연산

- 기본 연산자의 기호는 일반적인 프로그래밍 언어와 같다. (+ - * / %)
- Math 객체를 통해 복잡한 수치연산을 지원한다.

Math.pow(2,53)	9007199254740992 : 2의 53승	Math.E	e : 자연 로그 상수
Math.round(.6)	1.0 : 반올림	Math.sqrt(3)	3의 제곱근
Math.ceil(.6)	1.0 : 올림	Math.pow(3, 3)	3의 세제곱근
Math.floor(.6)	0.0 : 내림	Math.sin(0)	삼각함수
Math.abs(-5)	5 : 절대 값	Math.log(10)	자연로그10
Math.max(x,y,z)	가장 큰 인자를 반환한다.	Math.log(100)/Math.LN10	밑이 10인 로그 100
Math.min(x,y,z)	가장 작은 인자를 반환한다.	Math.log(512)/Math.LN2	밑이 2인 로그 512
Math.random()	0과 1.0 사이에 임의의 수 x를 반환한다.	Math.exp(3)	Math.E의 3 거듭제곱
Math.PI	파이 : 원주율		

- 산술 연산은 오버플로우, 언더플로우, 0으로 나누는 에러를 발생 시키지 않는다.

오버플로우	Infinity, -Infinity
언더플로우	0, -0
0으로 나누는 연산	오버플로우와 같음. 단 0을 0으로 나누면 NaN을 출력

- NaN의 독특한 기능 : 비교할 수 없다 (x == NaN 처럼 비교할 수 없음)
- x != x 이런 방식의 표현식 사용

숫자

✓ 이진 부동소수점과 반올림 오류

- 실수 연산을 할 때 근사 값으로 표현한다.
- IEEE-754 부동소수점 표현방식은 0.1같은 값을 정확하게 표현 할 수 없다
 - 현대적 프로그래밍 언어들의 문제

//부동소수점 오류 예시

```
var x = .3 - .2; // 0.3-0.2
```

```
var y = .2 - .1; // 0.2-0.1
```

```
x == y           // false:두 값은 같지 않다.
```

```
x == .1          // false:0.3-0.2는 0.1이 아니다.
```

```
y == .1          // true:0.2-0.1은 0.1과 같다.
```

- 따라서, 민감한 금융 계산에는 환산된 정수 값을 이용해야 한다.
 - ex) 달러 대신 정수 값인 센트로 단위변경

✓ 날짜와 시간

- Date() 생성자를 제공하며 Date 객체는 메서드를 가지고 있다.
- Date 객체는 원시타입이 아님

<참고>IEEE-754 - http://ko.wikipedia.org/wiki/IEEE_754

문자열

- ✓ 16비트 값들이 연속적으로 나열된 것
- ✓ 수정할 수 없는 유니코드 문자로 표현
- ✓ 0기반의 인덱싱을 사용(다른 프로그램과 동일)
- ✓ 문자 하나를 표현하는 문자형은 제공하지 않음
 - 길이가 1인 문자열을 사용해야 한다 char 같은 것은 없음

✓ 문자열 리터럴

- 작은 따옴표 또는 큰 따옴표로 둘러싸면 된다
- 큰 따옴표는 작은 따옴표로 둘러 싸인 문자열에 포함될 수 있고 작은따옴표 역시 큰따옴표로 싸여있는 문자열 안에 포함될 수 있다.
- 문자열 리터럴에 줄바꿈 문자가 필요한 경우, 이스케이프 문자 \n을 사용한다.

ECMAScript3	문자열 리터럴은 한 줄로 작성
ECMAScript5	\를 이용해 여러 줄로 작성가능

문자열

✓ 문자열 리터럴의 이스케이프 문자열

- 역슬래시(\)로 시작하는 이스케이프 시퀀스를 사용한다.

시퀀스	표현하는 문자
\0	널 문자 (\u0000)
\b	역스페이스 (\u0008)
\t	수평 탭 (\u0009)
\n	줄바꿈 문자 (\u000A)
\v	수직 탭 (\u000B)
\f	폼 피드(\u000C)
\r	캐리지 리턴 (\u000D)
\"	큰 따옴표 (\u0022)
\'	작은 따옴표 (\u0027)
\\	역슬래시 (\u005C)
\x	두 개의 16진수 숫자 XX에 의해 지정되는 Latin-1 문자
\u	네 개의 16진수 숫자 XXXX에 의해 지정되는 유니코드 문자

문자열

✓ 문자열 다루기

- 여러 문자열을 '+'를 이용하여 붙일 수 있음
- 문자열의 길이 : length 프로퍼티 사용
 - 그 외에도 문자열을 다루는 다양한 메서드가 존재함
- 자바스크립트에서 문자열은 변경되지 않음
 - replace() 와 toUpperCase() 같은 메서드는 새 문자열을 반환함

✓ 패턴 매칭

- 문자 패턴을 나타내는 객체를 생성하기 위해 RegExp() 생성자를 정의
- RegExp는 원시타입이 아니다. Date 객체처럼 유용한 API를 가지고 있는 특별한 종류의 객체이다.

```
//패턴 매칭 예시
var text = "testing: 1, 2, 3";
var pattern = /\d+/g;           //하나 이상의 모든 숫자와 일치
pattern.test(text);
text.search(pattern);
text.match(pattern);
text.replace(pattern, "#");
text.split(/\D+/);             //숫자가 아닌 문자열로 나눈다.
```

boolean, null과 undefined

✓ boolean 값

- true, false 중 하나의 값을 가짐
- 비교의 결과로 생성

AND 연산	&&
OR 연산	
NOT 연산	!

✓ null과 undefined

- null은 '객체가 아님'을 뜻하는 특수한 값
- undefined는 '값이 없음'을 나타내는 값
 - 값 자체가 없음
 - 초기화되어 있지 않거나, 존재하지 않는 값에 접근하려 할 때 얻는 변수의 값
- 시스템 레벨에서는 undefined, 일반적 프로그램 레벨에서는 null을 사용



3. 타입, 값, 변수

3.1 원시타입과 참조타입

3.2 원시타입

3.3 전역 객체와 레퍼객체

3.4 타입 변환

3.5 변수선언과 유효범위

- 전역객체

- 레퍼객체

전역 객체와 래퍼 객체

✓ 전역 객체

- 매우 중요한 용도로 사용되는 일반적인 자바스크립트 객체
- 최상위 코드에선 `this`를 이용해 참조

✓ 래퍼 객체

- 문자열, 숫자, 불리언의 프로퍼티에 접근할 때 생성되는 임시객체
- `String()`, `Number()`, `Boolean()` 생성자를 통해 생성할 수 있다.
- 문자열, 숫자, 불리언 값의 프로퍼티는 읽기전용이다.
- 래퍼 객체를 필요에 따라 기본 타입으로 변환할 수 있다.

<code>==</code>	값과 래퍼 객체를 동등하게 비교
<code>===</code>	값과 래퍼 객체를 구별

- `typeof` 연산자를 이용해 기본타입과 래퍼객체의 차이점을 볼 수 있음

```
var str = "Apple";  
str.concat(" is delicious"); // "Apple is delicious"
```

```
var str2 = new String("Apple");
```

```
str == str2; // true  
str === str2; // false
```

```
typeof str; // "string"  
typeof str2; // "object"
```



3. 타입, 값, 변수

- 3.1 원시타입과 참조타입
- 3.2 원시타입
- 3.3 전역 객체와 레퍼객체
- 3.4 타입 변환
- 3.5 변수선언과 유효범위

- 타입변환
- 변환과 동등비교, 명시적 변환
- 객체에서 원시타입으로 변환

타입 변환

- ✓ 자바스크립트는 타입에 대해 매우 유연하다.
- ✓ 어떤 타입의 값이든 전달할 수 있고, 그 값을 필요에 따라 변환할 수 있다.

```
var answer = 33;
console.log("The answer is " + answer); // "The answer is 33"
answer = "Welcome to the JavaScript world~!";
console.log(answer);

console.log("The answer is " + 33); // "The answer is 33"
console.log("33 is the answer");    // "33 is the answer"

console.log("37" - 7); // 30
console.log("37" + 7); // "377"

console.log(parseInt("123.45") + 1); // 124
console.log(parseFloat("123.45") + 1); // 124.45

console.log("1.1" + "1.1"); // "1.11.1"
console.log(("1.1") + ("1.1")); // 2.2
```

변환과 동등비교, 명시적 변환

✓ 변환과 동등비교

- 값의 타입을 유연하게 변환 시킬 수 있음
- 비교하기 전에 변환이 일어나고 그 후에 비교연산
- == 연산자를 사용한다.

✓ 명시적 변환

- 때때로 명시적 변환이 필요하고 코드를 깔끔하게 유지하기 위해 사용
- Boolean(), Number(), String(), Object() 함수를 사용한 명시적 형변환
- null, undefined 를 제외한 값은 toString() 메서드로 변환
- '+', '!' 등이 암시적 타입변환을 수행
- Number 클래스의 toString(출력할 진수) 메서드는 기수를 정하는 선택적 인자를 받음

정의하지 않음	10진수
2	2진수
8	"0" + 8진수
16	"0x" + 16진수

변환과 동등비교, 명시적 변환

✓ 숫자를 문자열로 변환하는 세 가지 메서드

- 결과 문자열 내의 나머지 숫자들을 적절히 반올림 함

toFixed()	소수점 이후에 정의된 수와 문자열로 숫자를 변환 지수표기법 사용 X
toExponential()	소수점 앞에 숫자 하나와 소수점 뒤에 지정된 만큼의 자릿수를 놓는 방식으로 숫자를 문자열로 변환 지수표기법 사용 O
toPrecision()	유효 자릿수가 숫자의 전체 정수부분을 표시할 정도로 크지 않다면 지수 표기법 사용 O

✓ parseInt(), parseFloat() 함수는 리터럴의 일부가 숫자가 아니어도 된다.

- Number() 보다 유연하다.
예) parseInt("33th") 는 33을 반환하지만 Number("33th")는 NaN이 된다.

✓ 객체에서 원시 타입으로 변환

- toString() : 해당 객체의 값을 문자열로 변환
- valueOf() : 해당 객체의 값을 원시타입으로 변환



3. 타입, 값, 변수

- 3.1 원시타입과 참조타입
- 3.2 원시타입
- 3.3 전역 객체와 레퍼객체
- 3.4 타입 변환
- 3.5 변수선언과 유효범위

- 변수 선언
- 변수의 유효범위
- 변수 끌어올림(Hoisting)
- 유효범위 체인

변수 선언

✓ 변수 선언

- 자바스크립트에서는 변수 선언 시 타입을 명시하지 않는다.
- var 키워드를 이용하여 변수를 선언한다.
- 여러 개의 변수를 한꺼번에 선언가능하고, 선언과 동시에 초기화할 수 있다.
(초기화 하지 않은 변수의 값은 undefined가 된다.)

✓ 반복하고 생략 가능한 변수 선언

- 변수를 반복적으로 선언할 수 있다.
- 표준모드에서 var선언 없이 변수에 값을 지정하면 동작한다
 - 나쁜 습관! 변수는 항상 선언 후 사용하자.

변수의 유효범위

- ✓ 프로그램에서 어떤 변수가 정의되어 있는 영역을 변수의 유효범위라 한다.
- ✓ 전역 유효범위에서는 var 구문을 사용하지 않고 전역 변수를 선언할 수 있다.
- ✓ 지역변수를 선언하려면 반드시 var 를 사용해야 한다.
- ✓ 함수의 유효범위와 끌어올림(Hoisting)
 - 타 프로그래밍 언어와 다르게 함수 안에 변수가 선언되면 함수 전체에 걸쳐 정의된 효과를 가짐
 - 함수 안의 모든 변수를 함수 맨 꼭대기로 끌어올린 것처럼 동작하는 특성

```
// 변수 끌어올림(hoisting) 예시
var scope = "global";
function f() {
  console.log(scope); // 출력되는 값은??
  var scope = "local";
  console.log(scope); // 출력되는 값은??
}

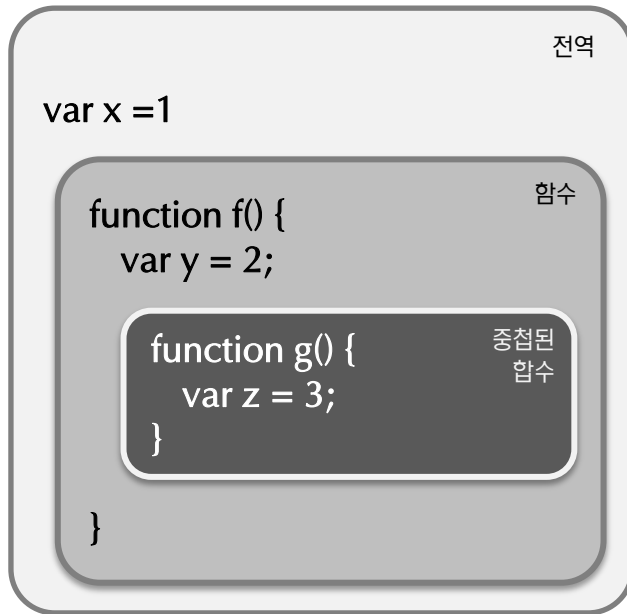
f();
```


변수의 유효범위

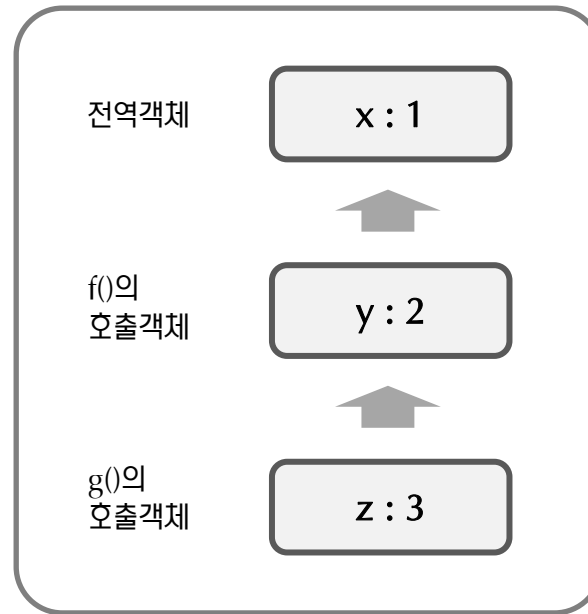
✓ 유효범위 체인

- 유효범위 안에서 변수를 정의하는 객체의 목록 또는 체인
- 유효범위체인의 범위는 전역 \in 함수 \in 중첩된 함수
- 유효범위 체인과 관련된 코드는 함수를 호출 할 때마다 달라지게 된다
- with() 구문과 함께 클로저를 이해하는데 중요한 개념이다

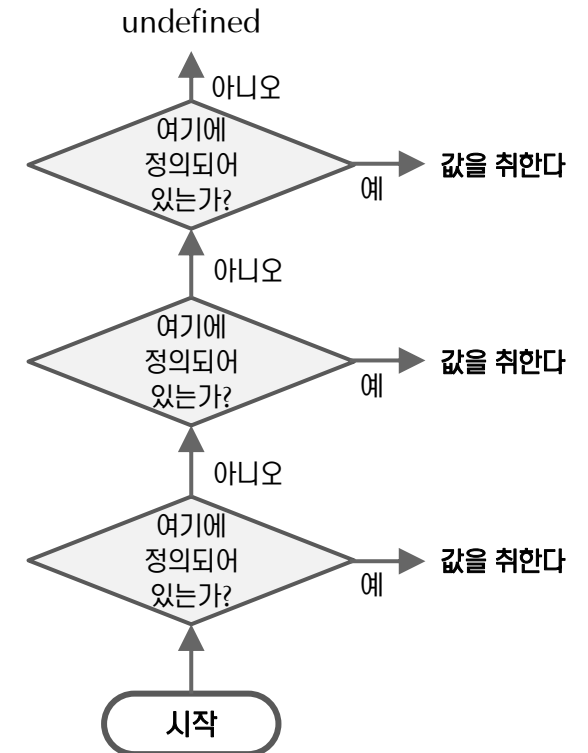
언어 구조적 유효범위



유효 범위 체인



변수 검색



변수의 유효범위

✓ 유효범위(Closure) 실습

```
var global_scope = "global";

function A(a_scope_param) {
  var a_scope = "a";

  var B = function() {
    var b_scope = "b";

    return function() {
      var c_scope = "c";
      return global_scope + a_scope + b_scope + c_scope + a_scope_param;
    };
  };

  return B();
}

console.log(A("첫번째")());
console.log(A("두번째")());
```



4. 표현식과 연산자

4.1 표현식

4.2 연산자

- 원시 표현식
- 객체와 배열의 초기화 표현식
- 함수 정의 표현식
- 프로퍼티 접근 표현식
- 호출 표현식
- 객체 생성 표현식

표현식

✓ 표현식

- 인터프리터에 의해 값으로 평가되는 자바스크립트 구문

✓ 원시 표현식

- 다른 표현식을 포함하지 않은 그 자체
- 상수, 리터럴 값, 특정 언어키워드, 변수 참조

✓ 객체와 배열의 초기화 표현식

- 새로운 객체나 배열을 값으로 하는 표현식
 - '객체리터럴', '배열리터럴'로 불림
- 특정 프로퍼티와 원소의 값을 지정하는 수많은 하위 표현식을 포함가능
 - 원시표현식이 아님

표현식

✓ 배열 초기화 표현식

```
[ ] // 빈 배열
[1+2, 2+3] // 3,5 두개의 원소를 가진 배열
var matrix = [[1,2,3], [4,5,6], [7,8,9]]; // 표현식 자체가 배열이 되어 중첩 배열을 만들
var sparseArray = [1,,,5]; // 쉽표사의의 값을 생략할 수 있음
```

✓ 객체초기화 표현식

- 사각괄호('[' ']') 대신 중괄호('{ ' }') 가 사용되고, 값 왼편에 프로퍼티 이름과 콜론(':')이 위치한다

```
var p = { x:2.3, y:-1.2 }; // 두개의 프로퍼티를 가진 객체
var q = {}; // 프로퍼티가 없는 빈 객체
```

✓ 함수 정의 표현식

- '함수 리터럴'이라 할 수 있음

```
var square = function(x) { return x * x; } //전달인자 이름과 함수 몸체로 이루어짐
```

표현식

✓ 프로퍼티 접근 표현식

- 객체의 프로퍼티나 배열의 원소 값으로 평가됨
- 접근법
 - 표현식 . 식별자
 - 표현식 [표현식]
- 점이나 대괄호 전에 표현식이 먼저 평가가 됨
- 평가된 값이 null 이나 undefined이면 프로퍼티를 갖지 않기 때문에 TypeError 발생
- 대괄호는 프로퍼티의 이름이 고정되어 있지 않고, 연산의 결과 그 자체로 사용될 수도 있음

//예제

```
var o = { x : 1, y : { z : 3 } }; // 간단한 객체 리터럴 예제
var a = [o, 4, [5,6]];           // 객체를 포함한 간단한 배열 리터럴 예제
o.x                               // o의 프로퍼티 x의 값
o.y.z                             // o.y의 프로퍼티 z의 값
o["x"]                             // o의 프로퍼티 x의 값
a[1]                               // a의 인덱스 1 위치에 있는 원소 값
a[2]["1"]                          // a[2]의 인덱스 1 위치에 있는 원소 값
a[0].x                             // a[0]의 프로퍼티 x의 값
```

표현식

✓ 호출 표현식

- 함수나 메서드를 호출하는 문법
- 여는 괄호로 시작해 쉼표(,)로 구분된 여러 호출 인자 목록이 올 수 있고, 닫는 괄호로 끝난다.

//표현식

```
f(0)           // f는 함수 표현식. 0은 인자 표현식  
Math.max(x,y,z) // Math는 함수, x,y,z 는 호출 인자
```

- 모든 호출 표현식은 한쌍의 괄호 () 와 괄호 안의 표현식으로 이루어짐
- 접근 표현식이 프로퍼티이면 호출 표현식은 메서드 호출
- 호출 시 함수의 몸체가 실행되는 동안 프로퍼티 접근 표현식이 가리키는 객체나 배열이 모두 this의 매개변수가 됨
 - 객체 지향적으로 프로그램을 작성할 수 있게 함
- 메서드 호출이 아닌 호출 표현식은 보통 전역객체를 this 키워드의 값으로 사용
 - ECMAScript 5 에서 '엄격한 모드'에서 사용시 undefined가 됨

표현식

✓ 객체 생성 표현식

- new 키워드가 앞에 붙음 (호출 표현식과 유사)
- 객체 생성시 생성된 객체는 객체 초기자 {} 에 의해 생성되는 객체와 동일

//표현식

new Object()

new Point(2,3)

//객체의 생성자 함수를 전달인자 없이 호출할 때, 다음과 같이 괄호를 생략할 수 있다.

new Object

new Date



4. 표현식과 연산자

4.1 표현식

4.2 연산자

- 연산자 살펴보기
- 산술 연산자
- 비드 단위 연산자
- 관계 연산자
- 논리 연산자
- 대입 연산자
- 평가 연산자
- 기타 연산자

연산자 [1/5]

연산자	수행되는 연산	결합 방향	피연산자 개수	피연산자 타입	반환
++	전치or후치 증가(단항 연산)	R	1	좌변 값	숫자
--	전치or후치 감소(단항 연산)	R	1	좌변 값	숫자
-	단항 마이너스(부정)	R	1	숫자	숫자
+	숫자로 변환	R	1	숫자	숫자
~	비트단위 NOT(단항 연산)	R	1	정수	정수
!	논리 NOT(단항 연산)	R	1	불리언	불리언
delete	프로퍼티를 제거	R	1	좌변 값	불리언
typeof	피연산자의 타입을 반환	R	1	타입 무방	문자열
void	undefined값을 반환	R	1	타입 무방	undefined
*, /, %	곱셈, 나눗셈, 나머지	L	2	숫자, 숫자	숫자
+, -	덧셈, 뺄셈	L	2	숫자, 숫자	숫자
+	문자열 이어 붙이기	L	2	문자열, 문자열	문자열

연산자 [2/5]

연산자	수행되는 연산	결합 방향	피연산자 개수	피연산자 타입	반환
<<	왼쪽으로 이동	L	2	숫자, 숫자	숫자
>>	부호 비트를 확장하면서 오른쪽으로 이동	L	2	숫자, 숫자	숫자
>>>	부호 비트 확장 없이 오른쪽으로 이동	L	2	숫자, 숫자	숫자
<, <=, >, >=	숫자를 비교	L	2	숫자, 숫자	불리언
<, <=, >, >=	문자열을 알파벳 순서로 비교	L	2	문자열, 문자열	불리언
instanceof	객체 타입 확인	L	2	객체, 생성자	불리언
in	프로퍼티가 존재하는지 확인	L	2	문자열, 객체	불리언
==	동등한지 비교	L	2	타입 무방	불리언
!=	동등하지 않은지 비교	L	2	타입 무방	불리언
===	일치하는지 비교	L	2	타입 무방	불리언
!==	일치하지 않는지 비교	L	2	타입 무방	불리언
&	비트단위 AND	L	2	숫자	숫자
^	비트단위 XOR	L	2	숫자	숫자
	비트단위 OR	L	2	숫자	숫자

연산자 (3/5)

연산자	수행되는 연산	결합 방향	피연산자 개수	피연산자 타입	반환
&&	논리 AND	L	2	타입 무방, 타입 무방	타입 무방
	논리 OR	L	2	타입 무방, 타입 무방	타입 무방
?:	조건부 연산자	R	3	불리언, 타입 무방, 타입 무방	타입 무방
=	할당	R	2	좌변 값, 타입 무방	타입 무방
*, /=, %=, +=, -=, &=, ^=, !=, <<=					
,	첫 번째 피연산자를 무시하고 두 번째 피연산자를 반환	L	2	타입 무방	타입 무방

연산자 [4/5]

✓ 피연산자 개수

- 피연산자 개수에 따라 n항 연산자라 부른다.

✓ 피연산자 반환 타입

- 대부분 반환 결과 타입이 정해져 있음
- 그러나 일부 연산자들은 피연산자의 타입에 따라 다름 (ex +, >, < 등)

✓ 좌변 값

- 할당 표현식의 좌변에 나타날 수 있는 표현식
- 자바스크립트에서는 변수, 객체 프로퍼티, 배열 원소

✓ 연산자 부수 효과

- 다시 평가될 때 기존의 값에 영향을 미치는 것
- 할당 연산자, 증가 연산자, 감소 연산자, delete 연산자

연산자 (5/5)

✓ 연산자 우선순위

- 우선적으로 연산하는 순서로 위의 표 위쪽부터 우선순위가 높다
- 프로퍼티 접근이나 호출 표현식은 위의 연산자들보다 항상 우선순위가 높다

✓ 연산자 결합방향

- L : 왼쪽에서 오른쪽
- R : 오른쪽에서 왼쪽

✓ 평가 순서

- 어떤 표현식이라도 평가 되면 부수 효과를 갖는다
- 평가될 표현식 중 하나라도 다른 표현식에 영향을 준다면 순서에 차이를 두어야 한다.

산술 표현식

✓ 덧셈 연산자 +

- 피연산자 숫자 값을 더하거나 피연산자 문자열을 붙인다
- 여러 숫자를 결합 시 연산자가 실행된 순서에 따라 연산 결과가 바뀜
 - 숫자 + 숫자 + 문자열 과 숫자 +(숫자 + 문자열) 의 값은 다름

✓ 단항 산술 연산자

- 하나의 피 연산자의 값을 수정해서 새 값으로 만듦
- 단항 덧셈 (+)
 - 피연산자를 숫자(or NaN)로 바꾼 후 값을 반환
- 단항 뺄셈 (-)
 - 피연산자를 가능 하면 숫자로 바꾸려고 시도 후 성공 시 결과 값의 부등호를 바꿈
- 증가 (++)
 - 1을 더하는 증가 연산을 한다
 - 앞에 붙으면 전치증가, 뒤에 붙으면 후치증가를 한다
- 감소 (--)
 - 1을 빼는 감소 연산을 한다
 - 앞에 붙으면 전치감소, 뒤에 붙으면 후치감소를 한다

비트 단위 연산자

✓ 이진수를 저 수준에서 조작하는데 사용

- 저 수준은 수준이 낮은 것이 아닌 시스템 쪽에 가깝다는 것을 의미한다

종류	부호	기능
비트단위 AND	&	개별 비트끼리 AND 연산을 한다
비트단위 OR		개별 비트끼리 OR 연산을 한다
비트단위 XOR	^	개별 비트끼리 배타적 불리언OR (XOR) 연산을 한다
비트단위 NOT	~	모든 비트를 반전 시킨다
왼쪽으로 이동	<<	첫 번째 피연산자의 모든 비트를 두번째 피연산에서 지정한 만큼 움직임
부호를 보존하면서 오른쪽으로 이동	>>	첫 번째 피연산자의 모든 비트를 두번째 피연산에서 지정한 만큼 움직임 0과 31 사이의 정수여야 함
0으로 채우면서 오른쪽으로 이동	>>>	>> 와 같다 차이점은, 왼쪽자리에 새로 들어오는 비트가 부호와 관계없이 무조건 0

관계 연산자

- ✓ 두 피연산자의 관계를 검사하여, 관계가 성립하면 true, 아니면 false를 반환
- ✓ 항상 불리언 값으로 평가가 됨

- ✓ 동등과 부등 연산자
 - == 동등
 - != 비 동등
 - === 일치
 - !== 비 일치

- ✓ 비교 연산자
 - < 더 작다
 - > 더 크다
 - <= 작거나 같다
 - >= 크거나 같다

논리 연산자

- ✓ 비교 연산자와 함께 사용되어 둘 이상의 변수가 관계되는 복잡한 표현식을 표현
- ✓ 10.1 논리 AND (&&)
 - 모두 true 이면 true
- ✓ 논리 OR (||)
 - 둘 중 하나라도 true 이면 true
- ✓ 논리 NOT (!)
 - 단항 연산자로 피연산자의 불리언 값을 반대로 한다.

대입 연산자

✓ = 연산자를 사용해 변수나 프로퍼티에 값을 할당한다

✓ 연산을 동반하는 할당

- +=, -=, *=, /=, %=
<<=, >>=, >>>=, &=, ^=
- 앞의 연산을 한 후 할당
 - ex) $a = a \text{ op(Operator) } b$ 와 같음
 - op(Operator) 는 연산자들이다

평가 연산자

✓ 문자열을 자바스크립트 코드로 해석하고 이를 평가한 결과를 값으로 출력

✓ 전역 함수 eval() 이 수행

✓ eval()

- 하나의 전달인자를 가짐
- 문자열이 아니면 넘긴 값을 반환
- 문자열을 전달하면 해석 해서 코드로 변환
- 코드해석에 문제가 있으면 syntaxError 발행

✓ 전역 eval()

- ECMAScript3 표준 : eval 이 다른 이름으로 호출되면 에러발생
- ECMAScript5 표준 : 'direct eval' 'eval' 아닌 다른 이름으로 eval()을 사용하는 예약어와 비슷
- 필수적인 인수로 최상위 내장 기능 함수
- 문자열로 구성된 JavaScript 문장을 직접 실행시키는데 유용하다

기타 연산자

✓ 조건부 연산자 (? :)

- 유일한 3항 연산자이다
- if문과 비슷하지만 좀더 편리하고 간결한 문법을 제공한다
 - ex) $A ? B : C$ 는 A가 참이면 B 거짓이면 C

✓ typeof 연산자

- 단일 피연산자 앞에 위치하는 단항 연산자
- 값은 피연산자의 데이터 타입을 가리키는 문자열
피연산자 값이 null 이면 "object"를 반환

✓ delete 연산자

- 단항 연산자로 피연산자로 지정된 객체 프로퍼티, 배열 원소 또는 변수의 삭제를 시도
- 삭제되서 존재하지 않는 객체 여부를 검사하려면 in 연산자를 활용
- 배열을 삭제하면 빈자리로 남는다
- 피연산자로 좌변값이 오지 않으면 true 반환, 성공적으로 삭제시 true 반환

연산자

✓ in 연산자

- 좌변 문자열, 우변 객체나 배열을 받음
- 좌변 값이 우변 객체의 프로퍼티 이름에 해당하면 true

✓ instanceof 연산자

- 좌변 객체, 우변 객체클래스 이름
- 좌변의 객체가 우변 클래스의 인스턴스일 경우 true

```
//예
var d = new Date();
d instanceof Date      // true
d instanceof Number    // false

var a = [1, 2, 3];
a instanceof Array;     // true
a instanceof Object;    // true
```

기타 연산자

✓ void 연산자

- 피 연산자는 어떤 타입도 될 수 있음
- 피 연산자를 무시하고 undefined를 반환
- javascript:URL 로 사용되나 대부분 onclick 이벤트 핸들러를 사용하기 때문에 사용빈도 낮음
-

✓ 쉼표(,) 연산자

- 이항 연산자로 왼쪽의 전달인자를 평가 하고 오른쪽의 전달인자를 평가한 후 반환
- 변수 선언이나 for 루프 안에 사용 된다



5. 구문

5.1 기본 구문

5.2 제어 구문

5.3 기타 구문

5.4 요약

- 구문
- 복합문과 빈 구문
- 선언문

구문

✓ 구문 (statements)

- 표현식은 어떤 값을 생성하기 위해 평가되지만 구문은 어떤 일을 하기 위해 실행된다.
- 자바스크립트에서 구문은 세미콜론(;)으로 끝난다.
- 자바스크립트 프로그램은 단순히 세미콜론(;)으로 구분된 구문들의 나열이다.

✓ 표현문 (expression statements)

- 가장 간단한 형태로는 부수효과가 있는 표현식
- 할당문, 함수호출 등이 있음

```
greeting = "Hello " + name;  
i += 3;  
counter++;  
delete o.x;  
alert(greeting);  
window.close();
```

복합문과 빈 구문

✓ 하나의 표현식 안에 여러 표현식을 합칠 때에는 쉼표(,)연산자를 사용한다.

✓ 구문블록

- 구문블록은 여러 구분을 중괄호로 감싼 것
 - 세미콜론으로 끝나지 않음
 - 들여쓰기를 해주는 것이 좋다 (가독성, 이해도)
 - 안에서 선언된 변수는 지역변수가 아닌 전역변수이다

✓ 빈 구문

- 자바스크립트 인터프리터는 빈 구문을 만나면 아무것도 실행하지 않는다
 - 종종 몸체가 비어있는 루프를 만들 때 유용
 - 임의로 빈 구문 사용시 코드에 고의로 사용했다는 설명을 주석으로 표시

```
;
for(i = 0; i < a.length; a[i++] = 0) ; //배열 a를 초기화한다.
```

선언문

✓ 선언문은 식별자를 정의, 변수나 함수를 생성하는 중요한 역할을 담당한다.

✓ var, function 은 삭제할 수 없는 변수를 만든다

✓ var

- 변수를 정의하는 데 쓰임
- 쉼표로 구분되어 여러 개를 한번에 선언 가능
- 초기화 하지 않으면 undefined 값을 가짐

✓ function

- 함수를 정의하는 데 쓰임
- 유일한 이름을 가짐
- 선언문 문법

```
function 함수이름(전달인자1, 전달인자2, ... , 전달인자n) {  
    구문  
}
```



5. 구문

5.1 기본 구문

5.2 제어 구문

5.3 기타 구문

5.4 요약

- 조건문
- 반복문
- 점프문

조건문 (1/2)

- ✓ 조건문은 특정 표현식의 값에 따라 구문을 실행시키거나 건너뛴다.
- ✓ if 문
 - 단순히 어떤 결정을 내리거나, 조건에 따라 좀더 정교하게 구문들을 실행한다.
 - 표현식이 true 인 경우 실행한다.
- ✓ else if 문
 - 조건문을 평가한 결과에 따라 여러 개 중 하나의 코드를 실행한다.

```
if (표현식)  
    구문
```

또는

```
if (표현식) {  
    구문  
} else if (표현식) {  
    구문  
} else {  
    구문  
}
```

조건문 (2/2)

✓ switch 문
switch(구문) {
 표현식
}

✓ 표현식의 모양

```
switch (구문) {  
    case 1:  
        //코드블록 1실행  
        break;  
    case 2:  
        //코드블록 2실행  
        break;  
    case 3:  
        //코드블록 3실행  
        break;  
    default:  
        //코드블록 4실행  
        break;  
}
```

반복문 (1/2)

✓ 반복문에는 while, do/while, for, for-in 문이 있다.

✓ while 문

- 조건식이 true인 동안 구문을 반복적으로 실행한다.

```
while (조건식) {  
    구문  
}
```

✓ do/while

- 적어도 한번은 루프 몸체(구문)를 실행하기 위해 사용한다.
- 끝에 세미콜론이 붙는다.

```
do {  
    구문  
} while (조건식);
```

반복문 (2/2)

✓ for 문

- 주어진 조건식이 true인 동안 반복문을 실행한다.
- 세미콜론으로 구분된 세가지 표현식을 가진다.
- 주로 배열의 요소를 하나씩 꺼내볼 때 사용한다.
- for(;;) == while(true)

```
for (변수 초기화; 조건식; 변수 업데이트) {  
    명령문  
}
```

✓ for-in 문

- 일반적인 for 문과는 달리 객체의 프로퍼티를 차례대로 제공한다.

```
// 객체 o의 모든 프로퍼티명을 배열 a에 담는 예제  
var o = {x:1, y:2, z:3};  
var a = [];  
var i = 0;  
for ( a[i++] in o);  
  
// 실행결과 a : ["x", "y", "z"]
```

```
for (변수 in 객체) {  
    명령문  
}
```


점프문

- ✓ 자바스크립트 인터프리터는 점프문을 만나면 특정 위치로 건너뛴다.
- ✓ 레이블
 - 프로그램의 진행 흐름을 제어하기 위해 사용하는 인식표
 - 어떤 구문에라도 그 앞에 식별자 이름과 콜론을 넣음으로 레이블을 붙일 수 있다.
- ✓ break
 - 반복하던 행위를 끝나치고 구문 밖으로 나온다
 - 루프나 switch 문 내부에서만 적법하다
 - break 키워드와 레이블 이름 사이에는 줄 바꿈을 할 수 없다.
 - 자바스크립트가 생략된 세미콜론을 자동으로 넣어주기 때문
 - break; 또는 break 레이블이름;

레이블 이름 :
구문
구문

점프문

✓ continue 문

- break 문과 유사하지만 루프를 빠져 나오지 않고 새로운 반복을 시작한다
- 항상 루프의 몸체 내부에서 사용해야 한다.
- continue; 또는 continue 레이블이름;

✓ return 문

- 함수호출 표현식의 값, 즉 함수에서 반환되는 값을 지정하는 데 쓰인다
- 오직 함수 몸체 내부에서만 나타날 수 있음
- return 키워드와 레이블 이름 사이에는 줄 바꿈을 할 수 없다.
 - 자바스크립트가 생략된 세미콜론을 자동으로 넣어주기 때문
- return 표현식;

✓ throw 문

- 예외적인 상황이나 오류를 발생시킬 때 사용한다.
- 예외가 발생하면 자바스크립트 인터프리터는 프로그램 실행을 즉시 중단하고 가장 가까운 예외처리기로 넘어간다
- 예외처리기를 찾을 수 없을 경우 해당 예외는 에러로 취급되고 사용자에게 보고된다
- throw 표현식;

점프문

✓ try / catch / finally 문

- 예외처리 기법
- try : 단순히 예외가 발생할 수 있는 코드 블록을 정의
- catch : try 블록 내부에서 예외가 발생할 경우 호출
- finally : try 블록에 일어난 상황과 관계없이 항상 실행이 보장 되어야 할 뒷정리용 코드가 포함
- try 블록은 catch, finally 둘 중 하나 이상의 블록과 함께 사용되어야 한다

```
try{  
    // 예외가 발생할 가능성이 있는 구문  
}  
catch (e) {  
    //try 블록에서 예외가 발생되어야만 실행 되는 부분  
    //throw 를 사용해서 예외를 다시 발생시킬 수도 있다  
}  
finally {  
    // try 블록에 일어난 일과 관계없이 try 블록이 종료되면 실행  
}
```



5. 구문

5.1 기본 구문

5.2 제어 구문

5.3 기타 구문

5.4 요약

- with 문
- debugger 문
- "use strict"

기타 구문

✓ 기타 구문

- with, debugger, use strict

✓ with

- 유효범위 체인을 임시로 변경하려 할 때 쓰임
- 유효범위 체인의 첫 번째에 '객체'를 추가 한 후 '구문'을 실행한 다음 유효범위체인을 '객체' 추가하기 전으로 되돌림
- 엄격한 모드에서 사용 할 수 없고 with 문 사용시 현저하게 느려지기 때문에 사용을 자제해야 한다
- 주로 깊이 중첩된 객체 계층 구조를 좀더 쉽게 다루기 위해 사용
with (객체)
구문

✓ debugger

- 디버거가 실행 중일 때 자바스크립트 구현체는 해당위치에서 정의된 코드 디버깅을 수행
- 중단점과 같이 동작

기타 구문

✓ "use strict"

- 지시어 다음에 오는 코드들이 엄격한 모드를 따르게 하기 위해서 사용한다.
- ECMAScript5 에서 처음 소개됨
- 지시어지만 구문에 가깝다
- 일반적인 구문과의 차이점
 - 키워드 목록에 포함되지 않음
- 엄격한 모드와 일반모드의 차이점 (중요 3가지)
 - with 문은 엄격한 모드에서 사용할 수 없다
 - 모든 변수는 반드시 선언 되어야 한다
 - 함수가 메서드가 아닌 함수로 호출 될 때 this 의 값은 undefined가 된다.



5. 구문

5.1 기본 구문

5.2 제어 구문

5.3 기타 구문

5.4 요약

- 구문 요약

구문 요약 (1/2)

구문	문법	용도
break	break 레이블;	가장안쪽의 루프, switch문 또는 '레이블'로 명명된 구문에서 빠져나온다
case	case 표현식;	switch문 내부의 구문에 레이블을 붙인다
continue	continue 레이블;	가장 안쪽의 루프, 또는 '레이블'로 명명된 루프를 재시작한다
debugger	debugger;	디버거 중단점
default	default;	switch문에서 디폴트 구문에 레이블을 붙인다
do/while	do 구문 while(표현식);	while 루프를 만드는 다른 방법
empty	;	아무 일도 안함
for	for(초기화; 테스트; 증가)구문	편리하게 쓸 수 있는 루프
for/in	for(변수 in 객체)구문	객체에 속한 프로퍼티들을 열거한다
function	function 이름(전달인자,...){ 구문}	'이름'이라는 함수를 선언한다
if/else	if(표현식){구문1} else{구문2}	구문1 또는 구문2를 실행한다
label	레이블 : 구문	'구문'에 '레이블'이라는 이름을 붙인다
return	return 표현식;	함수에서 값을 반환한다

구문 요약 [2/2]

구문	문법	용도
switch	switch(표현식){구문}	case 또는 default: 레이블이 붙은 구문들로 다중 분기
throw	throw 표현식;	예외를 발생시킨다
try	try {구문} catch(식별자){구문} finally{구문}	예외를 잡아낸다
use strict	"use strict";	스크립트나 함수를 엄격한 모드로 제한 시킨다
var	var 이름 =값 , ... ;	하나 이상 변수의 선언과 초기화
while	while (표현식) 구문	기본적인 루프 생성문
with	with (객체) 구문	유효범위 체인의 확장 (엄격한 모드에서 사용 불가)



6. 함수

6.1 함수의 정의와 호출

6.2 값으로서의 함수

6.3 네임스페이스로서의 함수

6.4 클로저

6.5 함수 프로퍼티, 메서드, 생성자

- 함수 개요
- 함수 정의하기
- 함수 호출하기
- 함수 매개변수와 전달인자

함수 개요

✓ 함수(function)란?

- 한 번 정의하면 몇 번이든 실행할 수 있고 호출 할 수 있는 자바스크립트 코드블록

✓ 함수는 일급객체

- 자바스크립트에서 함수는 객체이고 프로그램에 의해 처리될 수 있다.
- 자바스크립트는 함수를 변수에 할당 할 수 있다.
- 함수를 다른 함수로 전달 할 수 있다.

✓ 메서드

- 어떤 객체의 프로퍼티로 저장된 자바스크립트 함수

✓ 생성자

- 새로 생성된 객체를 초기화 하는데 쓰이는 함수

✓ 함수는 다른 함수 내에 중첩하여 정의할 수 있고, 변수 유효범위에 속한 어떤 변수에도 접근 가능하다 (클로저)

함수 정의하기 (1/2)

✓ 함수 정의하기

- 함수선언 : function 키워드와 함수이름 식별자를 정의한다.
- 매개변수 : 함수의 실행구문에 전달하기 위한 쉽표로 구분된 0개 혹은 임의 개의 식별자들과 이 식별자들을 둘러싼 한 쌍의 괄호
- 실행구문 : 0개 혹은 임의 개수의 자바스크립트 구문을 포함하는 한 쌍의 중괄호

```
function 함수이름(매개변수1, 매개변수2, ... ,매개변수n) {  
    //구문  
}
```

- * 함수 이름을 짧게 쓰기보다는 함수의 의미 및 기능을 잘 설명할 수 있는 이름으로 선택해야 함

함수 정의하기 (2/2)

✓ 중첩 함수

- 자바스크립트에서 함수는 다른 함수와 중첩될 수 있다
- 변수접근범위 : 중첩된 함수는 해당 함수가 속한 함수의 매개변수와 변수에 접근할 수 있다

```
function hypotenuse(a,b) {  
  function square(x) { return x * x; }  
  return Math.sqrt(square(a) + square(b));  
}
```

함수 호출하기 (1/3)

✓ 함수 호출하기

- 함수를 정의했어도 함수 몸체의 자바스크립트 코드는 함수를 호출하지 않으면 실행되지 않는다
- 자바스크립트 함수 호출법
 - 일반적인 함수 형태
 - 메서드 형태
 - 생성자
 - 해당 함수의 call()과 apply() 메서드를 통한 간접적 방식

✓ 함수 호출

- 순서
 - 각 전달인자 표현식(괄호 사이의 값들)이 평가
 - 평가 결과가 전달인자가 된다
 - 전달인자 값들은 함수정의에 지정한 매개변수와 대응된다
 - 함수 몸체에서 매개변수에 대한 참조는 해당 전달인자 값을 표현한다
- 반환 값
 - return 문에 값이 있으면 그 값을 반환하고, 아니면 undefined를 반환한다

함수 호출하기 (2/3)

✓ 메서드 호출

- 함수 : f , 객체 : o , 메서드 : m 라고 한다면 $o.m = f$; 로 정의 할 수 있다
- 객체 o 에 메서드 $m()$ 을 정의한 다음에 $o.m()$; 로 호출 할 수 있다
 - m 이 2개의 인자를 받는다면 $o.m(x,y)$; 로 호출하면 된다
(여러 개를 인자로 받을 수 있다)
- 메서드 호출과 함수호출은 호출 컨텍스트가 다르다.
 - 메서드 호출의 경우 $this$ 는 객체 자기 자신을 나타내는 반면 함수는 전역공간을 참조한다.

```
var calculator = {
  operand1: 1,
  operand2: 1,
  add: function() {
    //자기의 객체를 참조하기 위해 this 키워드 사용
    this.result = this.operand1 + this.operand2;
  }
};
calculator.add(); // add 메서드 호출
calculator.result // 2
```


함수 호출하기 (3/3)

✓ 생성자 호출

- 함수나 메서드 호출 앞에 new 키워드가 있으면 생성자 호출이다
- 한 쌍의 빈 괄호()는 생략 가능하다
- 새로 생성된 객체를 this 키워드로 참조할 수 있다

```
var o = new Object(); // 생성자 호출, 빈 괄호는 생략 가능
```

✓ 간접 호출 (메서드 빌려쓰기)

- call(), apply() 메서드를 이용하여 간접적으로 호출할 수 있다.
- 호출 시에 실행 컨텍스트가 되는 객체를 전달한다. 전달된 객체는 해당 함수 내에서 this로 참조된다.
 - 어떤 함수든지 특정 객체의 메서드로 호출할 수 있다
- call() 과 apply() 차이점
 - call()은 함수의 전달인자들을 나열한다. ex) someFunc.call(contextObj, arg1, arg2, ...);
 - apply()는 함수의 값 배열을 전달인자로 사용한다. ex) someFunc.apply(contextObj, [arg1, arg2, ...]);

함수 매개변수와 전달인자 (1/3)

✓ 함수 매개변수(parameters)

- 매개변수란 함수의 정의부분에 나열된 변수를 나타낸다.
- 함수 정의 시 매개변수에 대한 형식을 명시하지 않는다.
 - 주석을 통해 매개변수의 형식을 명시하도록 한다.

✓ 함수 전달인자(arguments)

- 전달인자는 함수를 호출할 때 전달되는 실제 값을 의미한다.
- 함수 호출 시 전달된 인자 값에 대한 형식 검사는 수행되지 않는다

✓ 생략 가능한 매개변수

- 생략 가능한 전달인자가 있는 경우, 매개변수 목록의 제일 뒤쪽에 정의한다.
- 함수를 정의 할 때 생략할 수 있음을 */*optional*/* 주석을 통해 강조한다.

```
// 주어진 객체의 모든 프로퍼티명을 배열에 담아 리턴한다.  
// 두번째 인자로 배열이 전달되는 경우 해당 배열에 추가한다. (생략가능)  
function getPropertyNames(o, /*optional*/ a) {  
    if ( a === undefined ) a = []; // a에 대한 전달인자가 없을 때 새로운 배열 생성  
    for(var property in o) a.push(property);  
    return a;  
}  
  
var a = getPropertyNames(o); // 전달인자 한 개. 새 배열에 o의 프로퍼티명을 넣음  
getPropertyNames(p, a);      // 전달인자 두 개. p의 프로퍼티명을 a에 추가
```

함수 매개변수와 전달인자 (2/3)

✓ 가변길이 전달인자 목록 : arguments 객체

- arguments 객체는 함수의 전달인자에 대응하는 유사 배열 객체이다.
- callee와 caller 속성
 - caller 속성은 현재 수행 중인 함수를 호출한 함수를 가져온다.
 - callee 속성은 현재 수행 중인 함수를 가리킨다. 익명 함수를 재귀적으로 호출하는 데 유용하다.

```
function add() {  
    var sum = 0;  
    for (var i = 0; i < arguments.length; i++) {  
        sum += arguments[i];  
    }  
    return sum;  
}  
  
function test() {  
    var sum1 = add(1, 2, 3);  
    var sum2 = add(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);  
}
```

함수 매개변수와 전달인자 (3/3)

✓ 객체의 속성을 전달인자로 사용하기

- 전달인자의 순서에 상관없이 이름/값의 쌍으로 호출

```
//전달인자를 순서에 상관없이 이름/값의 쌍으로 함수에 전달하기
function easyCopy(args) {
    testcopy(args.from,
              args.from_start || 0, //기본 값 0을 설정
              args.to_start || 0,
              args.length);
}

//easycopy() 사용법
var a = [1,2,3,4] , b = [];
easyCopy( { from: a, to: b, length: 4});
```



6. 함수

6.1 함수의 정의와 호출

6.2 값으로서의 함수

6.3 네임스페이스로서의 함수

6.4 클로저

6.5 함수 프로퍼티, 메서드, 생성자

- 값으로서의 함수

값으로서의 함수 [1/2]

- ✓ 자바스크립트 함수는 문법 뿐만 아니라 값이기도 하다. (함수는 일급객체)
 - 변수에 할당, 객체의 프로퍼티나 배열 요소로 저장, 다른 함수의 인자로 전달 등이 가능하다

//다른 함수의 인자로 전달하는 예(사칙연산)

//사칙연산 함수 정의

```
function add(x,y) { return x + y; }  
function subtract(x,y) { return x - y; }  
function multiply(x,y) { return x * y; }  
function divide(x,y) { return x / y; }
```

//정의한 함수 중 하나를 인자로 받아 두 개의 피연산자와 같이 호출

```
function operate(operator, operand1, operand2) {  
    return operator(operand1, operand2);  
}
```

//함수이용 (1 + 2) * (4 - 3)

```
var i = operate(multiply, operate(add, 1, 2), operate(subtract, 4, 3));
```

값으로서의 함수 [2/2]

✓ 자신만의 함수 프로퍼티 정의하기

- 정적 변수가 필요할 때는 함수의 프로퍼티를 사용하는 것이 좋다

```
//함수 객체의 카운터 프로퍼티를 초기화한다.  
//uniqueInteger 함수는 hoisting 되기 때문에 먼저할당 가능  
uniqueInteger.counter = 0;  
  
//다음반환값을 기억하기위해 자신의 프로퍼티사용  
function uniqueInteger() {  
    return uniqueInteger.counter++;  
}
```



6. 함수

6.1 함수의 정의와 호출

6.2 값으로서의 함수

6.3 네임스페이스로서의 함수

6.4 클로저

6.5 함수 프로퍼티, 메서드, 생성자

- 네임스페이스로서의 함수

네임스페이스로서의 함수

- ✓ 자바스크립트는 단일 코드블록 내에서만 유효한 변수를 정의하는 방법을 제공하지 않는다
 - 간단한 임시 네임스페이스처럼 작동하는 함수를 정의하는 방법

```
//방법1. 하나의 전역변수로 함수 정의
function myModule(){
    //모듈 코드
    //모듈에서 사용하는 어떤 변수이건 이 함수의 지역변수다
    //전역 네임스페이스를 어지럽히지 않는다
}
myModule(); //함수호출을 꼭 해주어야 한다!!

// 방법2. 익명 함수 정의와 즉시호출
(function() { //이름이 없는 표현식으로 위의 함수 재작성
    //모듈 코드

})(); //함수 리터럴을 끝내고 바로 호출함
```




6. 함수

6.1 함수의 정의와 호출

6.2 사용목적 별 함수 분류

6.3 클로저

6.4 함수 프로퍼티, 메서드, 생성자

- 클로저 (closure)

클로저 (closure)

- ✓ 클로저란 내부 함수가 외부 함수에 선언된 로컬 변수나 매개변수를 이용했는데 외부함수가 종료되어도 내부함수에서 사용한 변수 값은 내부 함수에서 계속 유지되는 특성을 말한다.

- 관점에 따라서는 모든 자바스크립트 함수가 클로저라고 볼 수 있음

```
function closureTest(a){  
    return function(b){  
        return a + b;  
    }  
}  
var test = closureTest(1);  
alert(test(2));
```

closureTest(1)을 호출한 순간 아래 영역에 var a = 10이 생성되고 function(b){ return a + b; }가 test에 반환된다. 그래서 test()를 사용할 수 있다.

test(2)를 호출하면 function(b){ return a+b; }이 기억하고 있던 a변수를 불러들이고 b의 값에 2가 대입된다
결과는 3이 나온다

✓ 주의점

- 클로저는 변수만 기억하고 값은 변수에 의해 참조될 뿐 기억되지 않는다
 - 반복문(for문)에서는 클로저를 사용하지 않는다.



6. 함수

6.1 함수의 정의와 호출

6.2 사용목적 별 함수 분류

6.3 클로저

6.4 함수 프로퍼티, 메서드, 생성자

- length 프로퍼티
- prototype 프로퍼티
- call과 apply 메서드
- toString 메서드
- Function 생성자

함수 프로퍼티, 메서드, 생성자

✓ length 프로퍼티

- 몸체 내에서의 arguments.length는 실제로 전달된 인자의 개수
- 함수 자체의 length 프로퍼티는 함수를 정의할 때 명시한 인자 개수를 반환

```
function check(args) {  
    var actual = args.length;           //인자의 실제 개수  
    var expected = args.callee.length; //인자의 요구 개수  
    if (actual !== expected)            //두 값이 다르면 예외 발생  
        throw Error("Expected " + expected + "args; got " + actual);  
}  
function f(x,y,z) {  
    check(arguments); //실제 인자 개수가 요구 개수와 같은지 검사한다  
    return x + y + z; //함수의 나머지 로직 수행  
}
```

✓ prototype 프로퍼티

- 모든 함수는 서로 다른 프로토타입을 가지고 있고, 프로토타입 객체를 참조한다
- 생성자 함수로 생성된 객체는 함수의 프로토타입 객체로부터 프로퍼티들을 상속받는다.

함수 프로퍼티, 메서드, 생성자

✓ call()과 apply() 메서드

- call()과 apply()는 어떤 함수를 간접적으로 호출할 수 있으며 특정함수를 다른 객체의 메서드인 것처럼 다룰 수 있게 한다.
- 첫 번째 인자는 호출되는 함수와 관련이 있는 객체
 - 실행 컨텍스트라고 하며 함수 몸체에서 this키워드의 값이 된다.

```
함수.call(객체, 전달할 인자, ...,전달할 인자);  
함수.apply(객체, [전달할 인자, ...,전달할 인자]);
```

✓ toString() 메서드

- 함수 선언 구문 다음에 나오는 문자열을 반환
- 실제 대부분은 toString() 메서드의 구현체들은 함수의 전체 소스코드를 반환

함수 프로퍼티, 메서드, 생성자

✓ Function() 생성자

- Function() 생성자를 통해서 함수를 정의할 수 있다
- 임의 개수의 문자열 인자를 요구 (마지막 인자는 함수 몸체에 대한 텍스트)
- 익명 함수를 생성한다 (함수 리터럴과 같음)
- 중요한 점
 - 동적으로 자바스크립트 함수를 생성하고 실행 시간에 컴파일 되는 것을 가능케 한다
 - 생성자가 호출될 때마다 함수 몸체를 분석하고 새로운 함수 객체를 생성한다
 - Function 생성자가 생성하는 함수는 어휘적 유효범위를 사용하지 않는다
 - . 언제나 최상위 레벨 함수로 컴파일
- new Function()으로 생성된 함수 객체를 호출하면 eval()함수를 호출하는 것과 같다(전역의 eval())
 - eval은 컨텍스트 및 스코프 관리가 어려워지므로 사용하지 않는 것이 좋다
 - new Function도 사용하지 않는 것이 좋다

```
var f = new Function("x","y","return x*y;"); //생성자를 사용한 정의
var f = function(x, y) { return x*y; }      //키워드를 사용한 정의
```




7. JavaScript 내장 객체

7.1 내장 객체

7.2 Number

7.3 String

7.4 Date

7.5 Array

7.6 Boolean

7.7 Math

7.8 RegExp

- JavaScript 내장 객체

- Object

JavaScript 내장 객체

- ✓ JavaScript에는 기본으로 내장된 객체가 있다.
- ✓ 내장 객체들
 - Object : 모든 JavaScript 객체의 부모
 - Number : 숫자를 담는 객체
 - String : 문자열을 담는 객체
 - Date : 날짜를 담는 객체
 - Array : 배열을 관리하는 객체
 - Boolean : 참/거짓 정보를 담는 객체
 - Math : 수학객체로서 수학기산에 필요한 상수와 함수를 가지는 객체
 - RegExp : 정규식 패턴 검색 결과에 관한 정보를 저장하는 내장 전역 객체

Object

✓ 최상위 객체인 Object

- 모든 자바스크립트 객체에 공통적인 기능을 제공한다.
- 어떠한 프로퍼티 값이든 정의할 수 있고 사용할 수 있다.
- 여러 메서드를 가지고 있다.

```
var obj = new Object();  
var obj2 = {};
```




7. JavaScript 내장 객체

7.1 내장 객체

7.2 Number

7.3 String

7.4 Date

7.5 Array

7.6 Boolean

7.7 Math

7.8 RegExp

- Number 객체

Number 객체

✓ Number 객체

- 자바스크립트에서 숫자는 하나의 객체 타입을 가지며, Number 객체는 64bit로 표현됨
- 소수점, 정수, 실수 등을 모두 Number 객체로 표현함
- 자바 스크립트 객체는 변수 타입을 별도로 지정하지 않음
- 정수 표현은 15자리 까지 정확한 표현할 수 있음
- 소수의 최대치는 17자리 이며 소수점 연산은 100% 정확하지 않음

✓ 속성

- MAX_VALUE : 숫자 표현으로 할 수 있는 최대 값
- MIN_VALUE : 숫자 표현으로 할 수 있는 최소 값
- NEGATIVE_INFINITY : 음수의 무한대
- POSITIVE_INFINITY : 무한대
- NaN : 숫자가 아님

✓ 메소드

- toExponential(x) : 숫자를 지수 표기법으로 출력
 - 파라미터 : 지수표기법에서 고정 소수점 자리 수이며 20을 넘을 수 없다.
- toFixed(x) : 고정 소수점 표시로 숫자를 나타냄
- toPrecision(x) : 지정된 자릿수를 사용하는 지수 표시나 고정 소수점 표시로 숫자를 나타냅니다.
 - 파라미터 : 최대 표현수
 - 정수부분의 표현이 파라미터 보다 적을 경우 지수로 표기한다.
- toString() : 숫자를 나타내는 문자열을 반환
- valueOf() : 지정한 숫자의 원시 값을 반환

```
console.log(Number.MAX_VALUE);  
console.log(Number.MIN_VALUE);
```

```
console.log(Number.NEGATIVE_INFINITY);  
console.log(Number.POSITIVE_INFINITY);
```

```
console.log(Number.POSITIVE_INFINITY == Number.MAX_VALUE*2); //true  
console.log(Number.NEGATIVE_INFINITY == Number.MAX_VALUE*-2); //true
```

```
console.log(isNaN(Number.MAX_VALUE)); // false;  
console.log(isNaN(Number.NEGATIVE_INFINITY)); // false;  
console.log(isNaN(Number.NaN)); // true
```


Number 객체

✓ 실습

```
<!DOCTYPE html>
<html>
<body>

<script>
console.log(115.1234567890123456789.toExponential());
console.log(115.12345678901234567890123456.toExponential(5));
console.log(115.12.toExponential(5));

console.log((new Number(123)).toPrecision());
console.log(123.5587.toPrecision(5));
console.log(1230000.5587.toPrecision(3));

console.log(1500.111.toFixed());
console.log(1500.123.toFixed(2));

var number = 0.1;
var s = number.toString();
console.log(s + " :length - " + s.length);

var num = 1234;
var valueOfData = num.valueOf();
console.log(num === valueOfData)

</script>
</body>
</html>
```



7. JavaScript 내장 객체

7.1 내장 객체

7.2 Number

7.3 String

7.4 Date

7.5 Array

7.6 Boolean

7.7 Math

7.8 RegExp

- String 객체

String 객체

✓ 문자열 표현 래퍼 객체

- 텍스트 문자열을 조작하고 서식을 지정 할 수 있다.
- 문자열 안에서 부분 문자열을 결정하고 위치를 지정할 수 있다.

✓ 속성

- length : 문자열의 길이 값

✓ 메소드

- charAt() : 지정한 인덱스에 해당하는 문자를 반환
- charCodeAt() : 지정한 위치에 있는 문자의 유니코드 값을 반환
- concat() : 둘 이상의 문자열을 연결한 문자열을 반환
- fromCharCode() : 여러 유니코드 문자 값에서 문자열을 반환
- indexOf() : 처음으로 나오는 부분 문자열의 위치를 반환
- match() : 정규식을 사용하여 일치하는 문자열을 검색하고 그 결과를 배열로 반환
- replace() : 정규식이나 검색 문자열을 사용하여 문자열에서 텍스트를 교체
- search() : 정규식 검색에서 첫 번째로 일치하는 부분 문자열을 찾음
- slice() : 문자열의 일정 부분을 반환
- split() : 지정된 구분 기호를 사용하여 문자열을 부분 문자열로 분할하고 배열로 반환
- substr() : 지정한 위치에서 시작하고 지정한 길이인 부분 문자열을 가져옴
- substring() : 개체 안의 지정된 위치에 있는 부분 문자열을 반환
- toLowerCase() : 영문자를 소문자로 변환
- toUpperCase() : 영문자를 대문자로 변환
- valueOf() : 문자열을 반환

String 객체

✓

```
(function (){
    var str = "ABCDEFGHJKLMNOPQRSTUVWXYZ";
    document.write(str.charAt(str.length - 1));
    document.write("<br>");
})();

(function (){
    str = "ABCDEFGHJKLMNOPQRSTUVWXYZ";
    document.write(str.charCodeAt(str.length - 1));
    document.write("<br>");
})();

(function (){
    var test = String.fromCharCode(112, 108, 97, 105, 110);
    document.write(test);
    document.write("<br>");
})();

(function (){
    var str1 = "ABCD"
    var str2 = "EFGH";
    var str3 = "1234";
    var str4 = "5678";
    document.write(str1.concat(str2, str3, str4));
    document.write("<br>");
})();
```

```
(function (){
    var str1 = "all good boys do fine";

    var slice1 = str1.slice(0);
    var slice2 = str1.slice(0, -1);
    var slice3 = str1.slice(4);
    var slice4 = str1.slice(4, 8);

    document.write(slice1 + "<br/>");
    document.write(slice2 + "<br/>");
    document.write(slice3 + "<br/>");
    document.write(slice4);
    document.write("<br />");
})();

(function (){
    var src = "azcafAJAC";
    var reg = /[a-c]/g;
    result = src.match(reg);

    for (var index = 0; index < result.length; index++)
    {
        document.write ("submatch " + index + ": " + result[index]);
        document.write("<br />");
    }
})();
```


String 객체



```
(function (){  
    var s = "The quick brown fox jumps over the lazy dog.";  
    var ss = s.substr(10, 5);  
    document.write("[ " + ss + " ] <br>");  
  
    ss = s.substr(10);  
    document.write("[ " + ss + " ] <br>");  
  
    ss = s.substr(10, -5);  
    document.write("[ " + ss + " ] <br>");  
    document.write("<br/>");  
    document.write("<br />");  
})();  
  
(function (){  
    var s = "The quick brown fox jumps over the lazy dog.";  
    var ss = s.substring(15, 10);  
    document.write(ss);  
    document.write("<br/>");  
})();
```

```
(function (){  
    var str1 = "This is a STRING.";  
    var str2 = str1.toLowerCase();  
    document.write(str2);  
  
    document.write("<br />");  
    document.write("<br />");  
})();  
  
(function (){  
    var str1 = "This is a STRING.";  
    var str2 = str1.toUpperCase();  
    document.write(str2);  
    document.write("<br />");  
})();
```



7. JavaScript 내장 객체

7.1 내장 객체

7.2 Number

7.3 String

7.4 Date

7.5 Array

7.6 Boolean

7.7 Math

7.8 RegExp

- Date 객체

Date 객체

✓ Date 객체

- 스크립트에서 시간을 담당하는 객체
- 밀리 초 단위로 특정 인스턴트 시간을 나타냄
- 인수의 값이 시간범위 보다 클 경우 해당 값을 수정해서 입력한다.
 - 150초 -> 2분 30초
- 나타낼 수 있는 날짜 범위는 1970년 1월 1일 전후로 약 285,616년
- Date 객체를 이용하면 날짜 비교하거나 경과 시간 등을 확인 할 수 있다.
- 일반적인 달력, 시계와 같은 작업들을 구현 할 수 있다.

Date 객체

✓ Date 사용법

```
<!DOCTYPE html>
<html>
<body>

<script>

var dateString = "Today's date is: ";

var newDate = new Date();

// Get the month, day, and year.
dateString += newDate.getFullYear() + "/";
dateString += (newDate.getMonth() + 1) + "/";
dateString += newDate.getDate();

document.write(dateString);

</script>
</body>
</html>
```


Date 객체

✓ 메소드

- 기본 날짜 제어 메소드
 - getDate() : 일을 반환 (1-31)
 - getDay() : 요일을 반환 (0-6)
 - getFullYear() : 년도를 반환 (4자리 숫자)
 - getHours() : 시간을 반환 (0-23)
 - getMilliseconds() : 밀리초를 반환 (0-999)
 - getMinutes() : 분을 반환 (0-59)
 - getMonth() : 월을 반환 (from 0-11)
 - getSeconds() : 초를 반환 (from 0-59)
 - getTime() : 1970년 1월 1일로 부터 대상날짜의 밀리 초를 반환
- 위 메소드에 대응하는 설정 메소드를 가지고 있다. 예: setDate()
- 사용되지 않는 메소드
 - getYear() : getFullYear() 함수로 대체
 - setYear() : setFullYear() 함수로 대체
 - toGMTString() : toUTCString 함수로 대체

Date 객체

✓ Date 객체 사용방법

- 현재 날짜 가져오기

```
var newDate = new Date();  
  
// Get the month, day, and year.  
dateString += newDate.getFullYear() + "/";  
dateString += (newDate.getMonth() + 1) + "/";  
dateString += newDate.getDate();  
  
document.write(dateString);
```

- 특정 날짜를 설정

```
var dt = new Date('2013-05-05');  
document.write(dt);
```

- 날짜 더하기/빼기

```
var myDate = new Date("1/1/1990");  
var dayOfMonth = myDate.getDate();  
myDate.setDate(dayOfMonth - 1);  
  
document.write(myDate);
```

Date 객체

✓ 문제 : 2014년 3월 달의 월요일 날짜를 모두 출력하시오.

- new Date("2014-03-01") 을 이용하여 날짜를 설정할 수 있다.
- getDay 함수를 이용하여 선택된 날짜의 요일을 알 수 있다.
- setDate 를 통해서 날짜를 변경할 수 있습니다.
- getDate로 날짜를 조회할 수 있습니다.
- getMonth는 0~11으로 달을 표현



7. JavaScript 내장 객체

7.1 내장 객체

7.2 Number

7.3 String

7.4 Date

7.5 Array

7.6 Boolean

7.7 Math

7.8 RegExp

- Array 객체

Array 객체

✓ 자바스크립트 배열

- 타입이 고정적이지 않음
 - 같은 배열에 있는 원소의 타입이 서로 다를 수 있습니다.
- 32비트 인덱스를 사용
- 크기가 동적이다.
 - 배열을 생성하거나 크기가 변경되어 다시 할당을 하더라도 배열 크기를 다시 선언할 필요가 없다.

Array 객체

✓ 배열 만들기

- 배열 리터럴 사용하기
 - 대괄호 안에 쉼표로 구분해 나열
 - 값으로는 리터럴 뿐 아니라 임의의 표현식도 사용할 수 있다.
 - 객체 리터럴 또는 다른 배열 리터럴을 포함할 수 있다.
 - 배열 리터럴에서 빠진 부분이 존재할 경우, 해당 부분의 원소 값은 undefined가 된다.
 - 제일 마지막에 쉼표를 추가할 수 있다.
- ex) var arr = [2, 3, 4, 5, , true, "a",];

Array 객체

- Array() 생성자 사용하기
 - 인자 없이 호출 (아무 원소도 없는 빈 배열, 리터럴과 동일)
 - `var a = new Array();`
 - 길이를 의미 하는 숫자 값을 인자로 주어 호출 (배열의 값과 인덱스 값이 존재하지 않는다)
 - `var a = new Array(44);`
 - 두 개 이상의 원소 값 or 숫자가 아닌 원소 값 하나를 명시적으로 지정
 - `var a = new Array(5, 4, 3, 2, 1, "test, test");`
- 배열의 생성자를 사용하는 것 보다 리터럴이 더 간편함
- 크기를 지정해주지 않으면 최대 크기로 설정($2^{32}-1$)

```
arrayObj = new Array()  
arrayObj = new Array([size])  
arrayObj = new Array([element0[, element1[, ...[, elementN]]]])
```

Array 객체

✓ 배열의 원소 읽고 쓰기

- 배열의 각 원소에 접근 할 시 [] 연산자 사용
 - 참조변수(배열 명)[음이 아닌 정수 값으로 평가되는 임의의 표현 식(인덱스 값)]
 - `val a = arr[4], var a = ["test"], a[a[i]] = a[0]`
- 자바스크립트는 사용자가 명시한 숫자 배열 인덱스를 문자열 형태로 바꿔서 속성 이름으로 사용한다.
- 배열의 인덱스와 객체의 속성이름을 올바르게 구별하는 것이 유용하다.
 - 모든 인덱스의 값은 속성 이름, 0과 $2^{32}-1$ 사이의 정수 값 속성이름만 인덱스
- 배열은 필요한 경우 length 속성 값을 자동으로 갱신한다.

Array 객체

✓ 희소배열

- 배열에 속한 원소의 위치가 연속적이지 않은 배열
- length 속성의 값은 원소의 개수보다 항상 큼
- 보통 배열보다 일반적으로 느리고, 메모리를 많이 사용하고, 원소를 찾는 시간이 오래 걸림

✓ 배열의 길이

- 모든 배열엔 length 속성이 있다.
 - 객체와 배열을 구분하는 중요한 특징
- 배열에 배열의 크기와 같거나 큰 값으로 인덱스 값을 설정 할 경우
 - $\text{length} = \text{인덱스 값} + 1$ 이다
- 배열은 항상 length 값과 같도록 구현되어 있다
 - `var test = {1,2,3,4,5};` 에서 `test.length = 0;` 을 하면
 - 모든 element를 삭제 하고 결과는 `{ }` 이다

Array 객체

✓ 배열에 원소 추가

- 직접 할당하기
 - 인덱스를 지정해 할당
- push() 메서드 사용하기
 - 배열의 끝에 원소를 추가
- unshift() 메서드 사용하기
 - 배열의 맨 앞에 원소를 추가, 기존원소들의 인덱스 값이 1씩 커짐

✓ 배열의 원소 삭제

- delete 로 삭제해도 배열의 길이는 변하지 않음
 - undefined 를 할당하는 것과 같음
- length 속성으로 배열의 끝에서부터 원소삭제가능
- pop() 메서드 사용하기
 - 배열의 length를 하나 줄이고 삭제된 값을 반환
- shift() 메서드 사용하기
 - 배열의 맨 앞에 원소를 삭제, 모든 원소의 인덱스 값을 하나씩 감소

Array 객체

✓ 배열 순회하기

- 가장 기본적인 방법은 for 문을 사용하는 것이다
- 배열의 length를 한번만 가져오도록 하여 루프를 최적화 시킨다.
- 배열의 원소를 사용하기 전에 null , undefined, 빈 원소가 있는지 확인하고 제외시킨다.

```
for(var i = 0, len = keys.length; i < len; i++){  
    if(!keys[i])  
        continue; // null , undefined, 빈 원소일 때 건너뛰  
}
```

Array 객체

✓ 다차원 배열

- 다차원 배열을 지원하지는 않지만 배열의 배열을 사용해 비슷하게 사용할 수 있다
- 배열 내의 배열에 있는 원소에 접근하는 법
 - [] 연산자를 두 번 사용하면 된다

//다차원 배열 만드는 법

```
var arr = new Array(10); // 10개의 행 만들기
```

```
for(var i = 0; i < arr.length; i++){
```

```
    arr[i] = new Array(10); // 각 행에 10개의 열을 만들기
```

```
}
```


Array 객체

✓ 배열 메서드

- ECMAScript 3의 Array.prototype 에는 배열을 다루는 여러 종류의 함수들을 정의하고 있음
- join()
 - 배열의 모든 원소들을 문자열로 변환하고 그 문자들을 이어 붙인 결과를 반환
 - 구분자는 쉼표(,)가 기본 값으로 사용
 - String.split() 메서드와 반대로 작동

```
var a = [1,2,3];  
a.join();           // "1,2,3"  
a.join(" ");        // "1 2 3"  
a.join("");         // "123"  
var b = new Array(10); //길이가 10인 배열  
b.join('-');         // 비어있으므로 사이의 9개의 -의 문자열이 생김
```

Array 객체

- reverse()
 - 배열의 원소 순서를 반대로 정렬하여 반환
 - 배열 안에서 직접 수행된다

```
var a = [1,2,3];  
a.reverse() // a는 [3,2,1]이 된다
```

- sort()
 - 배열 안의 원소들을 정렬하여 반환
 - 별도의 전달인자가 없을 경우 배열 안의 원소들을 알파벳순으로 정렬
 - 대소 문자 구별 없이 정렬 하고 싶으면 toLowerCase() 메서드를 사용

```
var a = new Array("banana", "cherry", "apple");  
a.sort(); // 알파벳순으로 정렬  
var b = a.join(", "); //"apple, banana, cherry"  
var c = [33,4,111,222];  
c.sort(function(a,b){ //오름차순 : 4, 33, 222, 1111  
    return a-b;  
});  
c.sort(function(a,b){ //내림차순 : 1111,222,33,4  
    return b-a;  
});
```

Array 객체

▪ concat()

- 기존 배열의 모든 원소에 concat() 메서드의 전달인자들을 추가한 새로운 배열 반환
- 배열의 원소 중에 배열이 있는 중첩 배열의 경우 원소를 꺼내지 않음

```
var a = [1,2,3];  
a.concat(4,5);      //[1,2,3,4,5]  
a.concat([4,5]);     //[1,2,3,4,5]  
a.concat([4,5],[6]); //[1,2,3,4,5,6]  
a.concat(4,[5,[6]]); //[1,2,3,4,5,[6]]
```

▪ slice()

- 부분 배열을 반환 (배열에서 잘라낸 원소들을 담은 새 배열)
- 처음과 끝을 알리는 전달인자 2개를 받음
- 전달인자가 음수면 우측(마지막 원소)으로 부터 앞쪽으로 전달인자 만큼 위치한 원소를 가리킴

```
var a = [1,2,3,4,5,6,7]  
a.slice(0,4);    //[1,2,3,4]  
a.slice(4);      //[5,6,7]  
a.slice(0,-2);   //[1,2,3,4,5]  
a.slice(-4,-2);  //[4,5]
```

Array 객체

- splice()
 - 배열의 원소를 삽입, 제거 시 범용으로 사용 가능
 - 처음과 끝을 알리는 전달인자 2개를 받음 3번째 전달인자 부터는 새롭게 삽입할 원소들을 지정
 . (처음, 처음으로부터 얼마나삭제할것인지, 새로삽입할 원소, 새로삽입할 원소,...)
 - 호출 시 배열 바로 수정
 - 삭제한 배열을 반환한다

```
var a = [1,2,3,4,5,6,7,8];  
a.splice(4);    //[5,6,7,8]반환 a는 [1,2,3,4]  
a.splice(1,2);  //[2,3]반환 a는 [1,4]  
a.splice(1,1);  //[4]반환 a는 [1]  
var b = [1,2,3,4,5];  
b.splice(2,0,'a','b'); //[ ]반환 b는 [1,2,'a','b',3,4,5]  
b.splice(2,2,[1,2],3); //['a','b']반환 b는 [1,2,[1,2],3,3,4,5]
```

Array 객체

- push(), pop()
 - 배열의 맨뒤에 원소를 추가하거나 제거할 수 있다.
 - 배열을 스택처럼 조작 가능하다. (FILO 스택을 구현할 수 있음)

```
var stack = [];  
stack.push(1,20); // [1,2] 2반환  
stack.pop(); // [1] 1반환  
stack.push(1,20); // [1,1,20] 3반환  
stack.pop(); // [1,1] 2반환  
stack.pop(); // [1] 1반환
```

- unshift(), shift()
 - 배열의 맨 앞에 원소를 추가하거나 제거하고 그에 따른 새로운 length 값을 반환한다.

```
var a = [];  
a.unshift(1); // [1] 1반환  
a.unshift(22,333); // [1,22,333] 3반환  
a.shift(); // [1,22] 333반환  
a.unshift(4,[5,6]); // [1,22,4,[5,6]] 4반환  
a.shift(); // [1,22,4] [5,6]반환  
a.shift(); // [1,22] 4반환
```


Array 객체

- toString() 과 toLocaleString()
 - toString() 메서드는 배열의 모든 원소를 문자열로 변환하고 쉼표(,)로 분리한 목록을 반환한다.
 - toLocaleString()은 toString의 지역화 버전으로 날짜를 문자열로 변환할 때 시스템의 날짜 표기법을 따른다.

```
[1, 2, 3].toString();           //결과 '1, 2, 3'  
["a", "b", "c"].toString();    //결과 'a, b, c'  
[1,['a','b'], 2].toString();    //결과 '1,a,b,2'
```

Array 객체

✓ 유사 배열 객체

- arguments
- 클라이언트 자바스크립트에서는 상당수의 DOM 메서드가 배열과 유사한 객체를 반환한다.
- 배열 메서드를 범용으로 정의한 이유는 배열과 유사한 객체에서도 작동하게 하기 위해서다.

✓ 문자열을 배열처럼 사용하기

- 대부분의 최근 브라우저에서는 문자열을 읽기 전용 배열처럼 다룬다.
 - [] 를 이용해 접근 가능
- 문자열에 범용 배열 메서드를 바로 사용할 수 있다.
- 문자열은 변하지 않는 값이라서 push(), sort(), reverse(), splice() 는 적용되지 않는다.



7. JavaScript 내장 객체

7.1 내장 객체

7.2 Number

7.3 String

7.4 Date

7.5 Array

7.6 Boolean

7.7 Math

7.8 RegExp

- Boolean 객체

Boolean 객체

✓ true / false 값을 표현하는 객체

- 일반적으로 조건문이나 반복문(if, while, for)에서 사용한다.
- 생성자를 사용할 경우 "true", "false" 문자열 혹은 0, 1의 숫자를 사용한다.
- 자동 캐스팅의 경우 숫자 0,1에 대해서 사용이 가능하다.

```
<!DOCTYPE html>
<html>
<body>

<script>

console.log( "1" == true);
console.log( " " == false);

console.log( 1 == true);
console.log( 0 == false);

console.log(undefined == false);
console.log(null == false);
console.log("false" == false);

</script>

</body>
</html>
```




7. JavaScript 내장 객체

7.1 내장 객체

7.2 Number

7.3 String

7.4 Date

7.5 Array

7.6 Boolean

7.7 Math

7.8 RegExp

- Math 객체
- Math 객체의 프로퍼티
- Math 객체의 함수

Math 객체

✓ Math Object

- 기본적인 계산 기능과 상수를 제공하는 내장 객체
- `new Math()` 같이 객체를 생성하여 사용할 수 없음

✓ 속성

- `Math.E` : 수학 상수 e . 자연 로그의 밑인 Euler의 상수
- `Math.LN2` : 2의 자연 로그
- `Math.LN10` : 10의 자연 로그
- `Math.LOG2E` : 밑이 2인 e 의 로그
- `Math.LOG10E` : 밑이 10인 e 의 로그
- `Math.PI` : π . 원주율
- `Math.SQRT1_2` : 0.5의 제곱근 값 또는 1을 2의 제곱근으로 나눈 값
- `Math.SQRT2` : 2의 제곱근

Math 객체

✓ Math Functions

- Math.abs : 숫자의 절대값을 반환
- Math.acos : 숫자의 아크코사인 값을 반환
- Math.asin : 숫자의 아크사인 값을 반환
- Math.atan : 숫자의 아크탄젠트 값을 반환
- Math.atan2 : X축에서 제공된 y 및 x 좌표로 표시된 점까지의 각도를 라디안 값으로 반환
- Math.ceil : 제공된 숫자 식보다 크거나 같은 가장 작은 정수를 반환
- Math.cos : 숫자의 코사인 값을 반환
- Math.exp : 승수로 거듭제곱하는 자연 로그의 밑인 e를 반환
- Math.floor : 제공된 숫자 식보다 작거나 같은 가장 큰 정수를 반환
- Math.log : 숫자의 자연 로그를 반환
- Math.max : 주어진 두 수식 중 큰 값을 반환
- Math.min : 주어진 두 수 중 작은 값을 반환
- Math.pow : 밑을 지정한 지수만큼 거듭제곱한 값을 반환
- Math.random : 0과 1 사이의 의사 난수 값을 반환
- Math.round : 지정된 수식을 반올림하여 가장 가까운 정수를 반환
- Math.sin : 숫자의 사인 값을 반환
- Math.sqrt : 숫자의 제곱근을 반환
- Math.tan : 숫자의 탄젠트 값을 반환

Math 객체

✓ Math 어디서 사용하나요?

- Html5의 Canvas 태그의 경우 타원 같은 그림을 그릴 경우 라디안 단위를 이용한 계산식을 사용해야 한다.
- round, floor 같은 함수는 수치 계산시 자주 사용된다.

```
<!DOCTYPE html>
<html>
<body>

<script>
document.write("floor: "+ Math.floor(1.699999999999) );
document.write("<br>");
document.write("round: "+ Math.round(0.6) );
document.write("<br>");
document.write("round: "+ Math.round(0.49) );
document.write("<br>");
document.write("random: "+ Math.random() );
document.write("<br>");
document.write("random + floor: "+ (Math.floor(Math.random() * 100 ) +1 )); // 1~100 random
document.write("<br>");
document.write("random + round: "+ (Math.round(Math.random() * 100 ) )); // 0~100 random
</script>

</body>
</html>
```



7. JavaScript 내장 객체

7.1 내장 객체

7.2 Number

7.3 String

7.4 Date

7.5 Array

7.6 Boolean

7.7 Math

7.8 RegExp

- RegExp 객체

RegExp 객체

✓ RegExp Object

- 정규식 패턴 검색 결과에 관한 정보를 저장하는 내장 전역 객체이다.
- 전역 RegExp 객체를 Regular Expression 객체와 혼동해서는 안됨
- RegExp : 패턴 검색 결과를 전역 저장하고 새로운 계산이 있을 경우 항상 업데이트 된다.
- Regular Expression : 자신의 인스턴스의 정보만 가지고 있다.

✓ 속성

- input : 정규식 검색에 사용한 문자열
- lastMatch : 마지막으로 일치하는 부분이 시작하는 문자
- lastParen : 마지막 괄호 안의 일치하는 값이 있을 경우 그 값을 반환
- leftContext : 처음부터 마지막으로 일치하는 문자열 앞까지의 문자
- rightContext : 마지막으로 일치하는 문자열 다음부터 검색한 문자열 끝까지의 문자
- \$1-\$9 : 패턴 일치 과정에서 찾아 가장 최근에 저장한 9개 부분을 반환

RegExp 객체

✓ 실습

```
<!DOCTYPE html>
<html>
<body>

<script>
//Create the regular expression pattern.
var re = new RegExp("d(b+)(d)","ig");
var str = "cdbBdbbsbdbdz";

// Perform the search.
var arr = re.exec(str);

// Print the output.
var s = ""
s += "$1: " + RegExp.$1 + "<br />";
s += "$2: " + RegExp.$2 + "<br />";
s += "$3: " + RegExp.$3 + "<br />";
s += "input: " + RegExp.input + "<br />";
s += "lastMatch: " + RegExp.lastMatch + "<br />";
s += "leftContext: " + RegExp.leftContext + "<br />";
s += "rightContext: " + RegExp.rightContext + "<br />";
s += "lastParen: " + RegExp.lastParen + "<br />";

document.write(s);
</script>

</body>
</html>
```

RegExp 객체

✓ Regular Expression 객체

- 문자열에서 문자 조합을 검색할 때 사용되는 패턴을 저장한다.
- 가장 최근에 검색한 정보는 전역 RegExp 객체에 저장된다.

✓ Properties

- global : 정규식에 사용되는 global 플래그(g)의 상태를 나타내는 부울 값을 반환
- ignoreCase : ignoreCase 플래그(i)의 상태를 나타내는 부울 값을 반환
- multiline : multiline 플래그(m)의 상태를 나타내는 부울 값을 반환
- source : 정규식 패턴의 텍스트 복사본을 반환

✓ Method

- compile : 빠른 실행을 위해 정규식을 내부 형식으로 컴파일을 실행함
- exec : 정규식 패턴을 사용하여 문자열을 검색하고 그 결과를 배열로 반환
- test : 문자열에 패턴이 있는지 여부를 나타내는 부울 값을 반환



8. Window 객체

8.1 Window 객체

8.2 Window 객체 활용

8.3 브라우저 객체

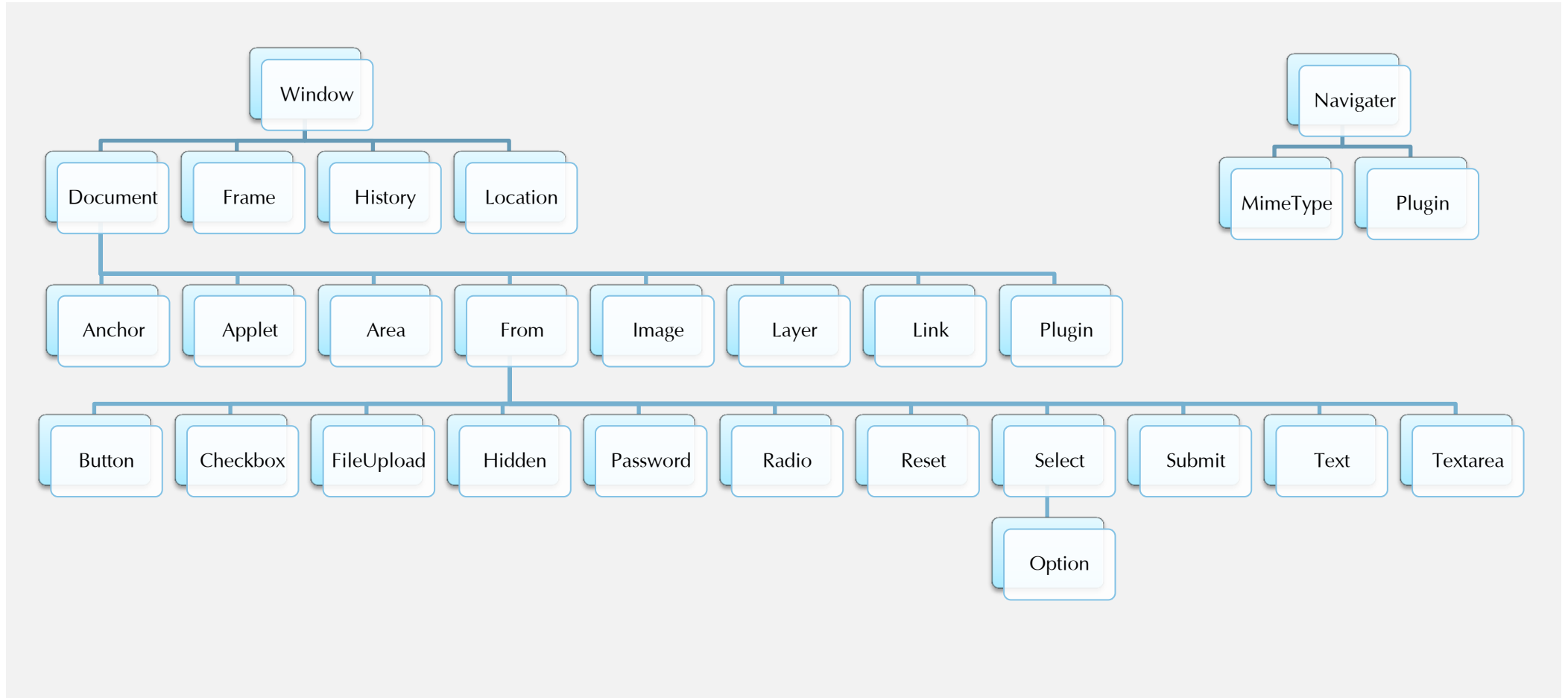
- Window 객체
- Window 객체 프로퍼티
- Window 객체 메서드
- Window 객체 이벤트 핸들러

Window 객체

✓ Window 객체

- 주 역할 : 클라이언트 측 자바 스크립트 프로그램의 전역 객체
- BOM(Browser Object Model)으로 불리기도 한다.

✓ 브라우저 내장객체



Window 객체 프로퍼티 (1/2)

프로퍼티	설명
status	브라우저의 상태바에 문자열을 출력하는 경우에 사용
defaultStatus	브라우저의 상태바에 초기 문자열을 설정
length	창안의 프레임 수
name	창 이름
self	현재 창 자신, window와 같음
window	현재 창 자신, self와 같음
parent	프레임에서 현재프레임의 상위프레임
top	현재프레임의 최상위프레임
opener	open()으로 열린 창에서 볼 때 자기를 연 창
document	document 오브젝트
frames	창안의 모든 프레임에 대한 배열정보
history	history 오브젝트 및 배열
location	location 오브젝트
closed	창이 닫혀 있는 상태
locationbar	location 바
menubar	창 메뉴 바

Window 객체 프로퍼티 (2/2)

프로퍼티	설명	
innerHeight	창 표시 영역의 높이(픽셀)	IE지원 안함
innerWidth	창 표시 영역의 너비(픽셀)	IE지원 안함
outerHeight	창 바깥쪽 둘레의 높이	IE지원 안함
outerWidth	창 바깥쪽 둘레의 너비	IE지원 안함
pageXOffset	현재 나타나는 페이지의	IE지원 안함
pageYOffset	현재 나타나는 페이지의 Y위치	IE지원 안함
Personalbar	창의 퍼스널 바	
Scrollbar	창의 스크롤 바	
statusbar	창의 상태 바	
toolbar	창의 툴 바	

Window 객체 메서드 (1/2)

메서드	설명	
alert()	경고용 대화상자를 보여줌	
clearTimeout()	setTimeout 메소드를 정지	
confirm()	확인, 취소를 선택할 수 있는 대화상자를 보여줌	
open()	새로운 창을 오픈	
prompt()	입력창이 있는 대화상자를 보여줌	
setTimeout()	일정 시간이 흐른후에 실행할 함수를 지정한다, millisecond 단위로 지정	
setInterval()	지정한 밀리초 주기마다 함수를 반복적으로 호출한다.	
eval()	문자열을 숫자로 바꿈	
toString()	오브젝트를 문자열로 바꿈	
blur()	focus를 이동	
focus()	focus를 줌	
scroll()	창을 스크롤 함	
valueOf()	오브젝트 값을 반환	
back()	한 단계 전 URL(이전화면)로 돌아감	IE지원 안함
find()	창안에 지정된 문자열이 있는지 확인 있다면 true 없으면 false	IE지원 안함
forward()	한 단계 뒤의 URL(다음화면)로 이동	IE지원 안함
home()	초기화 홈페이지로 이동	IE지원 안함

Window 객체 메서드 (2/2)

메서드	설명
moveby()	창을 상대적인 좌표로 이동. 수평방향과 수직방향의 이동량을 픽셀로 지정
moveto()	창을 절대적인 좌표로 이동. 창의 왼쪽 상단 모서리를 기준으로 픽셀을 지정
resizeby()	창의 크기를 상대적인 좌표로 재설정 밑변의 모서리를 기준으로 수평방향, 수직방향을 픽셀로 지정
resizeto()	창의 크기를 절대적인 좌표로 재설정 창 크기를 픽셀로 지정
scrollby()	창을 상대적인 좌표로 스크롤 창의 표시영역의 수평방향과 수직방향에 대해 픽셀로 지정
scrollto()	창을 절대적인 좌표를 스크롤 창의 왼쪽 상단 모서리를 기준으로 픽셀로 지정
stop()	불러오기를 중지 IE지원안함
captureEvents()	모든 타입의 이벤트를 판단
setInterval()	일정시간마다 지정된 처리를 반복
clearInterval()	setInterval 메소드의 정지
handleEvent()	이벤트 취급자를 정함
print()	화면에 있는 내용을 프린터로 출력
releaseEvent()	다른 계층의 이벤트로 이벤트를 넘김
routeEvent()	판단한 이벤트와 같은 계층의 이벤트
toSource()	오브젝트값을 문자열로 반환

Window 객체 이벤트 핸들러

이벤트 핸들러	설명
onBlur	브라우저가 포커스를 잃을 때 발생
onDragDrop	사용자가 다른곳에서 객체를 브라우저 안에 놓으려고 할 때 발생 IE지원 안함
onError	문서를 읽는 중에 에러가 생길 때 발생
onFocus	브라우저에 포커스를 얻을 때 발생
onLoad	문서를 읽을 때 발생
onMove	브라우저의 위치를 변경했을 때 발생 IE지원 안함
onResize	창의 크기를 변경했을 때 발생. IE지원 안함
onUnload	현재 문서를 지울려고 할 때 발생
onBlur	브라우저가 포커스를 잃을 때 발생
onDragDrop	사용자가 다른곳에서 객체를 브라우저 안에 놓으려고 할 때 발생 IE 지원 안함
onError	문서를 읽는 중에 에러가 생길 때 발생
onFocus	브라우저에 포커스를 얻을 때 발생
onLoad	문서를 읽을 때 발생
onMove	브라우저의 위치를 변경했을 때 발생 IE지원 안함
onResize	창의 크기를 변경했을 때 발생 IE지원 안함



8. Window 객체

8.1 Window 객체

8.2 Window 객체 활용

8.3 브라우저 객체

- open() 메서드
- 페이지 로딩 시 새창열기
- 클릭 시 새창열기
- 매개변수를 이용하기
- close() 메서드
- 새창 컨트롤 하기

새창열기

✓ 새창열기 open() 메서드

- window.open("문서url", "창이름", "창의 특성")
 - 첫째 인수 : 새 창으로 띄울 문서의 url
 - 둘째 인수 : 창 이름, 같은 경우엔 계속 창을 열 때 새로 열지 않고 이미 열린 창을 이용한다.
 - 셋째 인수 : 새로 열릴 창의 너비, 높이, 툴바, 상태바 등을 지정한다.

✓ 창의 특성

directories	yes no	익스플로러 연결도구모음, 익스플로러 전용
location	yes no	주소입력란
menubar	yes no	메뉴표시줄
scrollbars	yes no	스크롤바
status	yes no	상태표시줄
toolbar	yes no	도구모음
copyhistory	yes no	히스토리정보를 복사
resizable	yes no	창 크기 조절 가능여부
width	픽셀	창의 너비
height	픽셀	창의 높이

새창열기

✓ 페이지 로딩 시 새 창 열기

```
<script type="text/javascript">
function winOpen() {
    window.open("test.html", "test", "width=300,height=200,toolbar=no")
}
</script>

<body onLoad="winOpen()">
```

✓ 클릭 시 새 창 열기

```
<script type="text/javascript">
function winOpen() {
    window.open("test.html", "test", "width=300,height=200,toolbar=no")
}
</script>

<span onclick="winOpen()">클릭열기</span>
```

새창열기

✓ 클릭시 새창 열기 (링크에서)

```
<script type="text/javascript">
function winOpen() {
    window.open("test.html", "test", "width=300,height=200,toolbar=no")
}
</script>
<a href="javascript:winOpen()">링크열기</a>
```

✓ 매개변수를 이용하기

```
<script type="text/javascript">
function winOpen(url,winname,winhow) {
    window.open(url,winname,winhow)
}
</script>
<a href="javascript:winOpen('test.html','test','width=300,height=200,toolbar=no')">매개열기</a>
```

새창열기

✓ 새창닫기 close() 메서드

- window.close()

```
<script type="text/javascript">
function winClose() {
    window.close()
}
</script>
<a href= "javascript:winClose()"> 함수이용해서 닫기 </a>
<a href= "javascript>window.close()"> 메서드 이용 닫기 </a>
```

새창열기

- ✓ 새로 열린 창에서 연 창을 컨트롤하기

```
<script type="text/javascript">
// 창 닫기 전에 연 창의 폼요소에 값 넘기기
function winClose(data) {
    opener.form1.data=data
    self.close()
}
</script>
<a href= "javascript:winClose('값1')"> 선택1</a>
<a href= "javascript:winClose('값2')"> 선택2</a>

<script type="text/javascript">

// 창 닫기 전에 연 창을 리로드하기
function winClose() {
    opener.location.reload()
    self.close()
}
</script>
<a href="javascript:winClose()"> 함수이용해서 닫기 </a>
```


새창열기

✓ 새로 열린 창에서 크기 조절하기

- window.resizeTo (너비, 높이)

```
<script type="text/javascript">
// 페이지로딩시 크기 조절
function winSize() {
    window.resizeTo(300,200) // 너비,높이
}
</script>
<body onLoad="winSize()">
```

✓ 새로 열린 창에서 위치 조절하기

- window.moveTo (x좌표, y좌표)

```
<script type="text/javascript">
// 페이지로딩시 위치 조절
function winMove() {
    window.moveTo(200,200) // X,Y 좌표
}
</script>
<body onLoad="winMove()">
```



8. Window 객체

8.1 Window 객체

8.2 Window 객체 활용

8.3 브라우저 객체

- Navigator 객체
- Location 객체
- History 객체
- alert(), confirm(), prompt()

Navigator 객체

- ✓ Navigator 객체는 브라우저의 정보가 내장된 객체이다.
 - HTML5에서는 위치 정보를 알려 주는 역할도 담당한다.

```
<div id="example"></div>
```

```
<script>
```

```
txt = "<p>Browser CodeName: " + navigator.appCodeName + "</p>";  
txt+= "<p>Browser Name: " + navigator.appName + "</p>";  
txt+= "<p>Browser Version: " + navigator.appVersion + "</p>";  
txt+= "<p>Cookies Enabled: " + navigator.cookieEnabled + "</p>";  
txt+= "<p>Platform: " + navigator.platform + "</p>";  
txt+= "<p>User-agent header: " + navigator.userAgent + "</p>";
```

```
document.getElementById("example").innerHTML=txt;
```

```
</script>
```

Location 객체

- ✓ Location 객체를 이용하여 현재 페이지 주소(URL) 등을 알 수 있다.
- ✓ 또한 다른 페이지로 이동할 수 있다.

```
<!DOCTYPE html>
<html>
<body>
<script>

document.write(location.href);

</script>
</body>
```

History 객체

- ✓ History 객체는 브라우저의 페이지 히스토리를 담은 객체이다.
 - 사용자의 개인정보 보호를 위해 가장 기본적인 기능만 제공한다.
 - history.back() : 브라우저의 뒤로 가기 버튼과 동일한 행위를 한다.
 - history.forward() : 브라우저의 앞으로 가기 버튼과 동일한 행위를 한다.

```
<html>
<head>
<script>
function goBack()
{
    window.history.back();
}
function goForward()
{
    window.history.forward();
}
</script>
</head>
<body>

<input type="button" value="Back" onclick="goBack()">
<input type="button" value="Forward" onclick="goForward()">
</body>
</html>
```


Popup box

- ✓ alert() : 브라우저 경고창을 띄운다.
- ✓ confirm() : 브라우저로 확인/취소 선택창을 보여준다.
- ✓ prompt() : 브라우저로 입력 창을 발생시켜 사용자의 입력을 받는다.

```
<!DOCTYPE html>
<html>
<head>
<script>
function myAlert()
{
    alert("I am an alert box!");
}

function myConfirm()
{
    if(confirm("select button")){
        console.log("OK");
    }else{
        console.log("Cancel");
    }
}

function myPrompt()
{
    console.log(prompt("input text"));
}
</script>
```

```
</script>
</head>
<body>

<input type="button" onclick="myAlert()" value="Show alert box">
<input type="button" onclick="myConfirm()" value="Show confirm box">
<input type="button" onclick="myPrompt()" value="Show Prompt box">

</body>
</html>
```



9. DOM (Document Object Model)

9.1 DOM 개요

9.2 문서 계층 구조

9.3 DOM 제어

9.4 이벤트

- DOM 개요

DOM 개요

✓ DOM(Document Object Model) 개요

- HTML과 XML 문서의 내용을 조작하고 나타내는 기반 API
- 문서의 요소 집합이 DOM에서는 계층구조로 표현된다
- HTML태그나 요소를 나타내는 노드, 텍스트 문자열을 나타내는 노드를 포함한다

부모 노드 : 한 노드의 바로 위 노드

자식 노드 : 한 노드의 바로 아래 노드

형제 : 같은 부모 노드를 가진 레벨이 같은 노드

자손 : 한 노드 아래로 쪽 연결된 모든 노드의 묶음

조상 : 한 노드 위쪽으로 부모등의 위쪽의 모든 노드

```
<html>
  <head>
    <title>예제 문서</title>
  </head>
  <body>
    <h1>HTML 문서</h1>
    <p>이것은<i>예제</i>문서</p>
  </body>
</html>
```




9. DOM (Document Object Model)

9.1 DOM 개요

9.2 문서 계층 구조

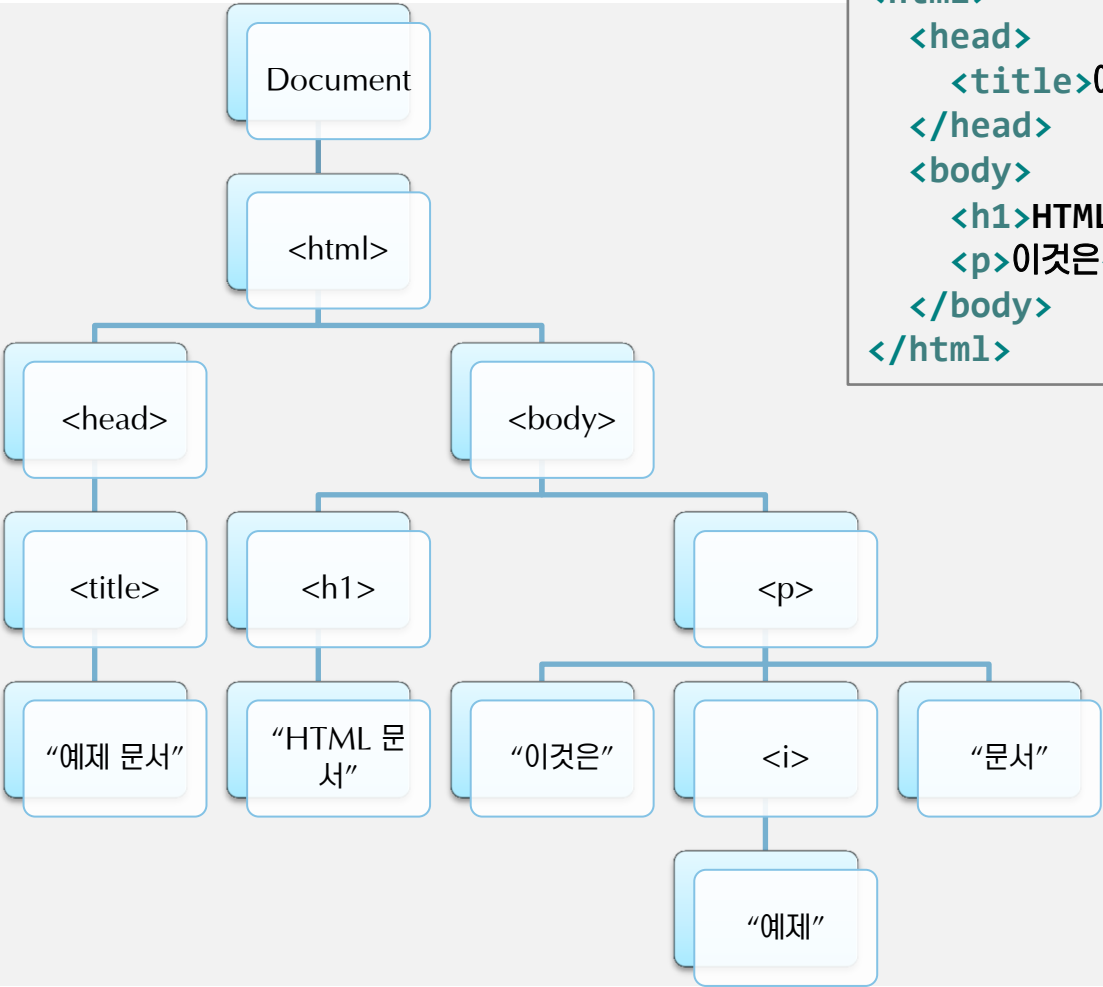
9.3 DOM 제어

9.4 이벤트

- HTML 문서의 계층 구조 표현
- 문서노드의 클래스 계층 구조

HTML 문서의 계층 구조 표현

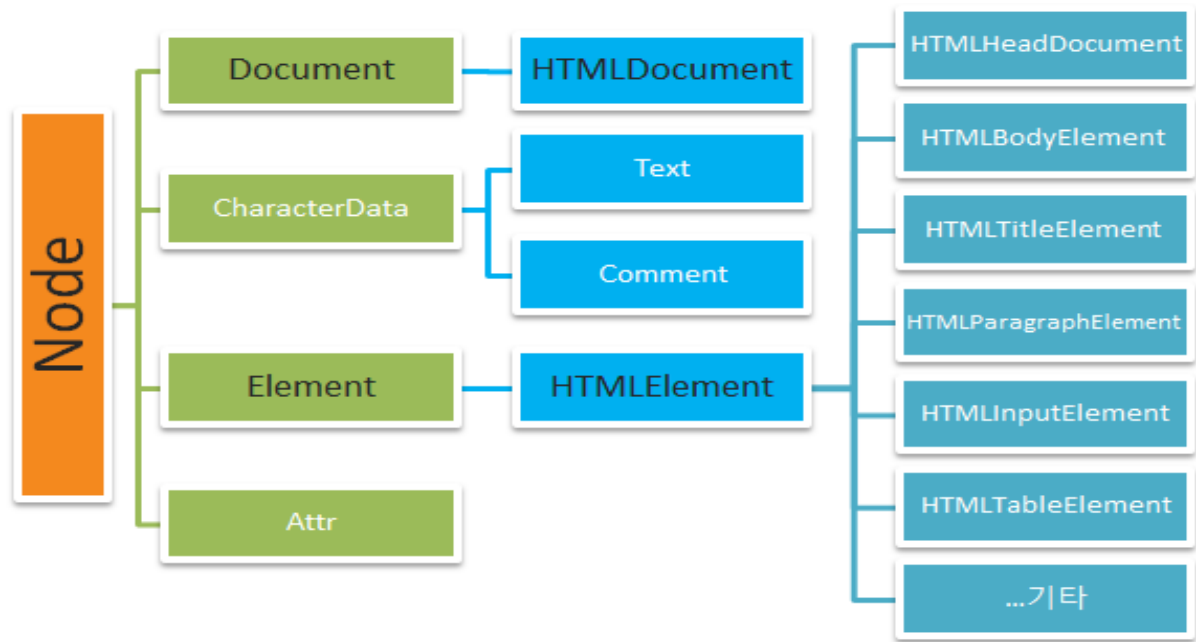
✓ HTML 문서의 계층 구조 표현



```
<html>
  <head>
    <title>예제 문서</title>
  </head>
  <body>
    <h1>HTML 문서</h1>
    <p>이것은<i>예제</i>문서</p>
  </body>
</html>
```


문서 노드의 클래스 계층 구조

- ✓ 제네릭타입 : Document, Element
 - Document 타입 : HTML 또는 XML 문서
 - Element 클래스 : 그 문서의 요소를 나타냄
- ✓ 일반타입 : HTMLDocument, HTMLElement
 - 제네릭타입의 서브 클래스
 - 좀더 구체적으로 HTML의 문서와 요소만을 말함
 - HTML 단일요소나 요소 집합의 속성에 해당하는 프로퍼티를 정의



* 중요 부분 검은색으로 표시



9. DOM (Document Object Model)

9.1 DOM 개요

9.2 문서 계층 구조

9.3 DOM 제어

9.4 이벤트

- DOM 제어하기

DOM 제어

✓ JavaScript를 이용하여 DOM을 제어할 수 있다.

- HTML Element 변경
- HTML Element의 속성 변경
- CSS 스타일 변경
- 페이지의 모든 이벤트 제어

✓ HTML Element 검색하기

- id, name, tag name 을 통해서 검색할 수 있다.
- CSS3 selector를 이용하여 검색할 수 있다.(HTML5에 추가된 내용)

DOM 제어

✓ HTML Element 검색하기 실습

```
<!DOCTYPE html>
<html>

<body>
  <span id="div1"> div1 content</span>
  <div> div2 content</div>
  <span name="div3"> div3 content</span>

  <script>
    var element1 = window.document.getElementById("div1");
    var element2 = window.document.getElementsByTagName("div")[0];
    var element3 = window.document.getElementsByName("div3")[0];

    console.log(element1.innerHTML);
    console.log(element2.innerHTML);
    console.log(element3.innerHTML);
  </script>
</body>

</html>
```

DOM 제어

✓ HTML 내용 변경하기 실습

```
<!DOCTYPE html>
<html>

<body>
  <span id="div1"> div1 content</span>

  <script>
    var element1 = window.document.getElementById("div1");
    element1.innerHTML = "new content";
  </script>
</body>

</html>|
```


DOM 제어

✓ HTML Style 변경 실습

```
<!DOCTYPE html>
<html>

<body>
  <span id="div1"> div1 content</span>

  <script>
    var element1 = window.document.getElementById("div1");
    element1.style.color = "red";
  </script>
</body>

</html>
```

```
<!DOCTYPE html>
<html>
<body>

  <h1 id="id1">My Heading 1</h1>
  <button type="button"
    onclick="document.getElementById('id1').style.color='blue'">
    Click Me!</button>

</body>
</html>
```



9. DOM (Document Object Model)

9.1 DOM 개요

9.2 문서 계층 구조

9.3 DOM 제어

9.4 이벤트

- DOM 이벤트
- Mouse 이벤트
- Keyboard 이벤트
- Frame/Object 이벤트
- Form 이벤트
- 버블링과 캡처링

DOM 이벤트

- ✓ JavaScript는 DOM에서 발생하는 이벤트에 반응하여 다양한 작업을 할 수 있다.
- ✓ 다음과 같은 상황에서 이벤트가 발생한다.
 - 사용자가 마우스를 클릭 하였을 때
 - 웹 페이지가 로딩 되었을 때
 - 이미지가 로딩 되었을 때
 - 마우스가 Element 사이로 움직일 때
 - Input 필드가 변경 되었을 때
 - Form 이 submit 되었을 때
 - 키보드가 눌러 졌을 때

Mouse 이벤트

✓ Mouse Events

- onclick : 마우스로 Element 를 클릭 했을 때
- ondblclick : 마우스로 Element를 더블 클릭했을 때
- onmouseup : 마우스로 Element에서 마우스버튼을 up 할 때
- onmousedown : 마우스로 Element에서 마우스 버튼을 down 할 때
- onmouseover : 마우스를 움직여서 Element 위로 올릴 때
- onmouseout : 마우스를 움직여서 Element 에서 벗어 날 때
- onmouseenter : 마우스를 움직여서 Element 밖에서 안으로 들어 올 때
- onmouseleave : 마우스를 움직여서 Element 안에서 밖으로 나갈 때

Mouse 이벤트

✓ 주요 Mouse Event 속성

- target : 이벤트가 발생한 요소
- screenX : 화면상에서 이벤트가 발생될 때 마우스 X좌표
- screenY : 화면상에서 이벤트가 발생될 때 마우스 Y좌표
- clientX : 화면상에서 이벤트가 발생될 때 Element를 기준으로 X좌표
- clientY : 화면상에서 이벤트가 발생될 때 Element를 기준으로 Y 좌표
- ctrlKey : Ctrl 키를 같이 눌렀는지 여부(boolean)
- shiftKey : Shift키를 같이 눌렀는지 여부(boolean)
- altKey : Alt 키를 같이 눌렀는지 여부(boolean)

Keyboard 이벤트

✓ Keyboard Events

- onkeydown : 키보드를 누르는 순간
- onkeypress : 키보드가 눌러 졌을 때
- onkeyup : 키보드를 키를 놓았을 때

Frame/Object 이벤트

✓ Frame/Object Events

- onabort : 이미지 등의 내용을 로딩하는 도중 취소 등으로 인해서 중단되었을 때
- onerror : 이미지 등의 내용을 로딩 중 오류가 발생 하였을 때
- onload : document, image, frame 등이 모두 로딩 되었을 때
- onresize : document, element 의 크기가 변경되었을 때
- onscroll : document, element 가 스크롤 되었을 때
- onunload : 페이지나 Frame등이 unloading 되었을 때

Form 이벤트

✓ Form Events

- onblur : Input과 같은 Element 등에서 입력 포커스가 다른 곳으로 이동할 때
- onchange : 입력 내용이 변경 되었을 때
- onfocus : Input과 같은 Element에 입력 포커스가 들어 올 때
- onreset : 입력폼이 리셋 될 때
- onselect : Input, Textarea 에서 입력값 중 일부가 마우스 등으로 선택될 때
- onsubmit : Form이 submit(전송) 될 때

Event 실습

✓ Event 실습

```
<!DOCTYPE html>
<html>
<head>
<script>
function changetext(id)
{
id.innerHTML="Oops!";
}
</script>
</head>
<body>
<h1 onclick="changetext(this)">Click on this text!</h1>
</body>
</html>
```

```
<!DOCTYPE html>
<html>
<body>

<input onkeydown="mOver(this)" onkeyup="mOut(this)" value="key down" />

<script>
function mOver(obj)
{
obj.value="Thank You"
}

function mOut(obj)
{
obj.value="key down"
}
</script>

</body>
</html>
```

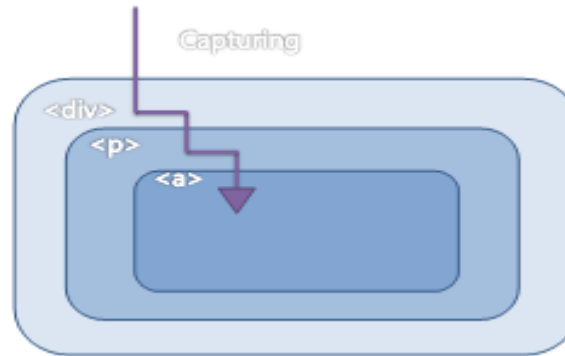
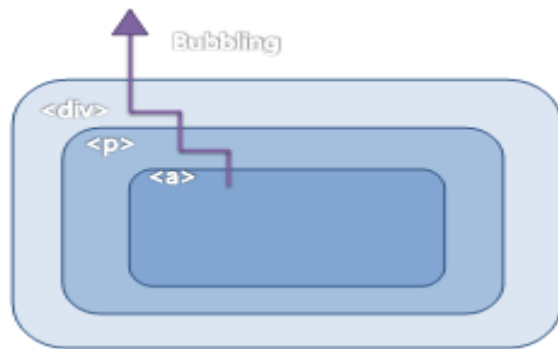
버블링과 캡처링

✓ 이벤트 버블링

- 이벤트 요소부터 이벤트 요소를 포함하고 있는 부모 요소까지 올라가며 이벤트를 검사
- 버블속성의 이벤트 핸들러가 있다면 실행

✓ 이벤트 캡처링

- 이벤트가 발생한 요소를 포함하는 부모 HTML 부터 이벤트 근원지인 자식요소까지 이벤트를 검사
- 캡처속성의 이벤트 핸들러가 있다면 실행시키면서 접근



✓ 제약사항

- 보통 기본 이벤트는 버블 속성이다
- W3C 표준에는 이벤트를 묶을 때 캡처 핸들러인지 버블 핸들러인지 지정할 수 있게 되어있다
- IE 계열은 캡처 이벤트를 지원하지 않는다.

토의

- ✓ 질의 응답
- ✓ 토론

감사합니다...

- ❖ 넥스트리소프트(주)
- ❖ 김현오 선임 (hyunohkim@nextree.co.kr)