Ch2. JavaScript

Introduction & basics

Jae-Ho Choi

CONTENTS

I .JAVA Script

1.1JavaScript 개요

- HTML5와 JavaScript
  - HTML5 에는 JavaScript 가 상당히 많은 분량을 차지하고 있다.
  - HTML5 의 로직처리를 JavaScript가 담당하고 있기 때문.
  - 이러한 JavaScript 개발에 대해 알아보고,
    데이터 전달을 위한 JSON에 대해 알아 보도록 한다.

1.1JavaScript 개요

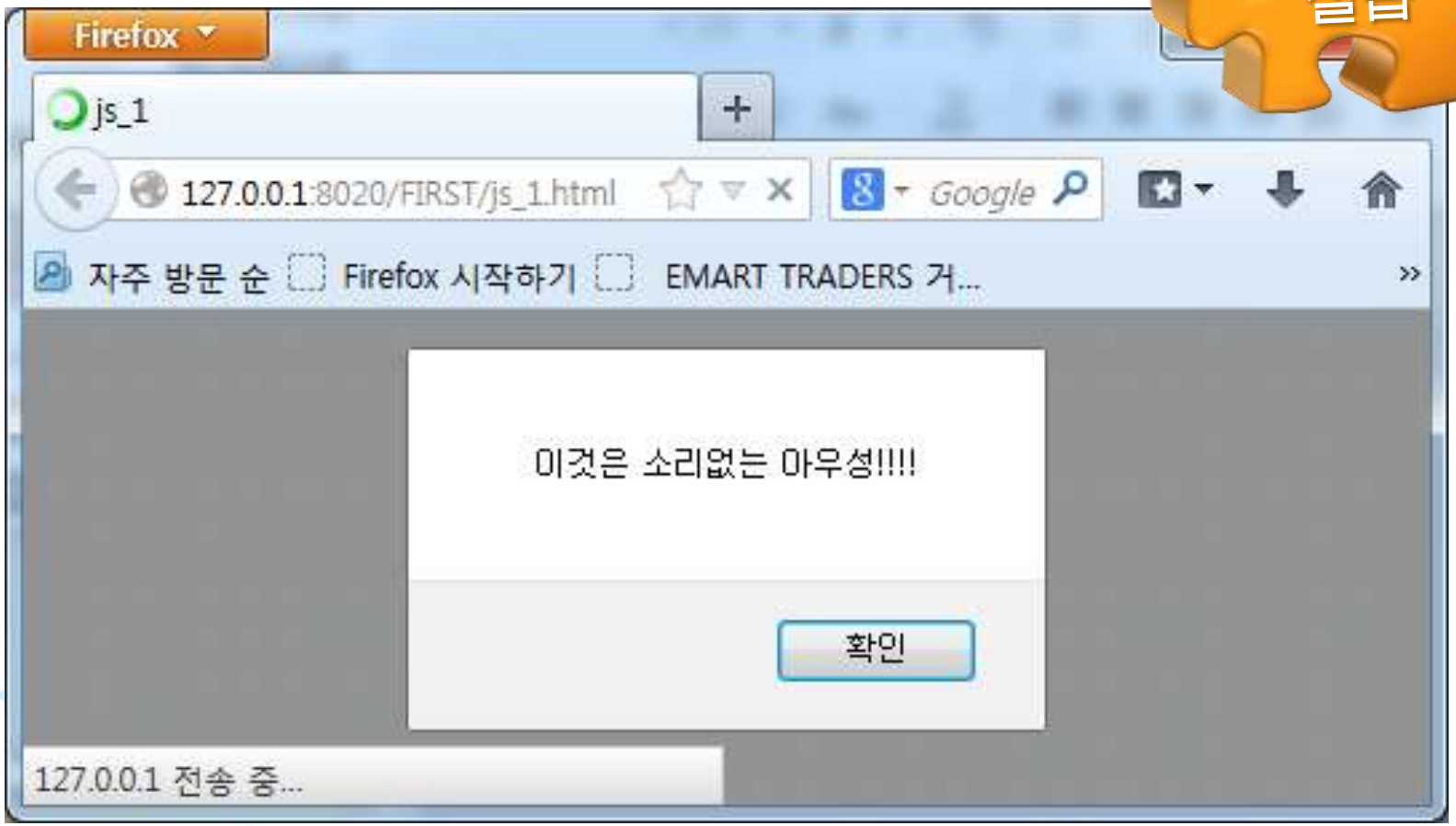
- Javascript란?
  - 웹 브라우저에서 사용하는 프로그래밍 언어
  - 넷스케이프 사의 브랜든 아이히(Brendan Eich)에 의해 모카라는 이름으로 만들어짐.
  - 이후 라이브 스크립트라는 이름으로 개발
  - 넷스케이프 사가 썬 마이크로시스템과 함께 자바스크립트라는 이름을 붙여 주고 본격적 발전

- 자바스크립트 코드 위치
  - 기본 페이지의 head 태그 사이에 script 태그 삽입

```
<!DOCTYPE html>
<html>
<head>
    <title></title>
    <script>

    </script>
</head>
<body>

</body>
</html>
```

1.1 JavaScript 개요

- 다음의 프로그램을 만들어 보자.



1.1JavaScript 개요

- 자바스크립트 작동
  - HTML 페이지의 각 태그는 웹 브라우저에 의해 순차적으로 실행
  - 웹 브라우저는 head 태그를 먼저 읽은 후 body 태그를 읽음
  - script 태그를 head 태그에 넣으면 body 태그를 읽기 전 실행
  - body 태그에 넣으면 head 태그를 실행한 이후에 실행

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8" />
        <title>index</title>
        <script>
            alert("이것은 소리없는 아우성!!");
        </script>
    </head>
    <body>
    </body>
</html>
```

1.2 식별자

- 식별자 개요
  - 자바스크립트에서 이름을 붙일 때 사용
  - 식별자의 예 - 변수명과 함수명
  - 식별자 생성시 규칙
    - ✓키워드를 사용하면 불가
    - ✓숫자로 시작하면 불가
    - ✓특수 문자는 _과 $만 허용
    - ✓공백 문자 포함 불가
    - ✓대소문자 구분

```
alpha
alpha10
_alpha
$alpha
AlPha
ALPHA
```

1.2 식별자

- 식별자 생성 규칙
  - 모든 언어 사용 가능하나 알파벳 사용이 개발자들 사이 관례
  - input, output 같은 의미 있는 단어 사용
  - 생성자 함수의 이름은 항상 대문자로 시작
  - 변수, 인스턴스, 함수, 메서드의 이름은 항상 소문자로 시작
  - 여러 단어로 이뤄진 식별자는 각 단어의 첫 글자를 대문자로.

- will out ⇨ willOut
- will return ⇨ willReturn
- i am a boy ⇨ iAmABoy

1.2 자바스크립트 주석 및 출력

- 자바스크립트 주석
  - // 를 사용해 한 줄 주석 표현
  - // 뒤의 문장은 실행되지 않음
  - /* 와 */을 사용해 여러 줄 주석 표현


```
<head>
  <meta charset="utf-8" />

  <title>JavaScript Test</title>
  <script>
      // 이 부분은 한줄 주석 입니다.
      /*
        alert("Hello World!!");
        alert("Hello World!!");
      */
  </script>
</head>
```

1.2 자바스크립트 주석 및 출력

- Javascript 출력
  - 가장 기본적인 출력 방법 - alert() 함수 사용
  - 웹 브라우저에 경고창 띄울 수 있음
  - 함수의 괄호 안에는 문자열 입력
  - console.log를 이용한 출력
  - alert() 함수의 사용 예

```
<head>
  <meta charset="utf-8" />

  <title>JavaScript Test</title>
  <script>
      alert("Hello World!!");
  </script>
</head>
```

1.3 문자열

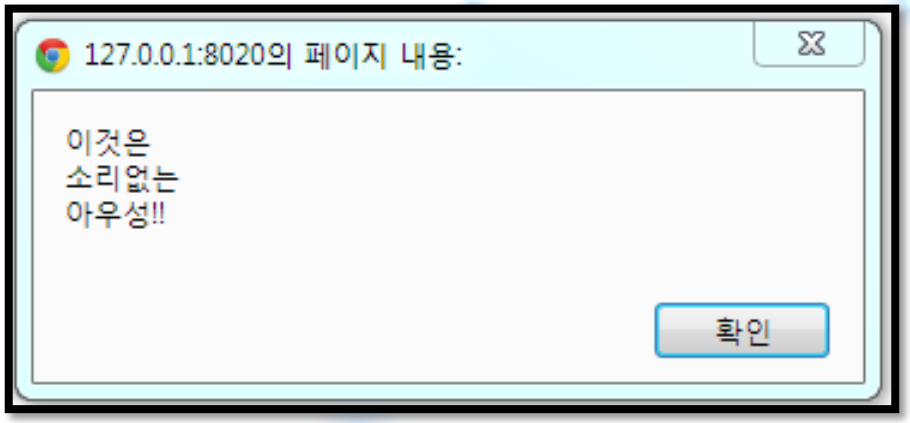
- 문자열이란?
  - 문자를 표현할 때 사용하는 자료의 형태
  - alert() 함수의 매개 변수로 쓰인 “Hello World!!” 와 같은 자료
  - 문자열을 만드는 방법
  - “동해물과 백두산이” (큰 따옴표)
  - ‘동해물과 백두산이’ (작은 따옴표)
  - 두 가지 중 어떤 방법으로 문자열을 만들어도 되지만, 일관되게 사용할 것
  - 문자열 안에 쓰는 따옴표
    - ✓ 내부에 작은 따옴표를 쓰고 싶으면 외부에 큰 따옴표
    - ✓ 내부에 큰 따옴표를 쓰고 싶으면 외부에 작은 따옴표

```
<head>
  <meta charset="utf-8" />

  <title>JavaScript Test</title>
  <script>
        alert("Hello 'World!!'");
  </script>
</head>
```

1.3 문자열

- Escape 문자열
  - 특수한 기능을 수행하는 문자
  - 문자 그대로 따옴표를 사용하고 싶다면? (\" 이용)
  - Escape 문자 \n은 문자열을 줄 바꿈할 때 사용


```
alert("이것은\n소리없는\n아우성!!");
```



1.3 문자열

- 예외적인 문자열 사용법
  - 연결 연산자 '+'
  - "가나다" + "라마" + "바사아" + "자차카타" + "파하"
  - "가나다라마바사아자차카타파하"
  - "가나다" + 10
  - "가나다10"
  - "원주율은 : " + 3.1415926535
  - "원주율은 : 3.1415926535"

1.4 변수와 연산자

- 숫자 자료형
  - 정수와 유리수의 구분 없이 숫자는 모두 숫자
- % 연산자
  - 좌변을 우변으로 나눈 나머지를 표시하는 연산자
- boolean(불리언)
  - 자바스크립트에서
    참과 거짓이라는 값을 표현할 때 사


```
alert(true);
alert(10 < 100);
alert("가" > "나" );
alert("이것은" > "소리없는");
alert("Hello" > "안녕");
alert(10%7);
alert(10 + 20 < 25);
alert("boolean값은 : " + true);
alert(100 + true);
alert(true > false);
```

1.4 변수와 연산자

- 자바 스크립트 논리연산자

연산자	설명
!	논리 부정 연산자
&&	논리곱 연산자
||	논리합 연산자

```
var tmp = 20;

alert(30 > tmp > 10);

alert((30 < tmp) && (tmp < 10));
```

1.4 변수와 연산자

- 변수
  - 값을 저장할 때 사용하는 식별자
  - 변수지만 숫자뿐만 아니라 모든 자료형 저장 가능
  - 변수를 사용하려면?
    1. var 식별자 이용 변수 선언
    2. 변수 초기화를 이용한 변수 선언


```
var variable1;

var variable2,variable3=100,variable4;

variable5 = 200;
```

1.4 변수와 연산자

- 변수의 Data Type
  - 자바스크립트에는 총 여섯 가지 자료형이 있음
  - 문자열, 숫자, 불리언, 함수, 객체, undefined
  - undefined 자료형
    - ✓선언되지 않거나 값이 할당되지 않은 변수


```
var variable1; // undefined
var stringVar = "Hello World";
var numberVar = 3.1415;
var booleanVar = true;
var myFunc = function() { };
var myObj = { };
```

1.4 변수와 연산자

- 복합 대입 연산자
  - 대입 연산자와 다른 연산자를 함께 사용하는 연

연산자	설명
+=	기존 변수의 값에 값을 더합니다.
-=	기존 변수의 값에 값을 뺍니다.
*=	기존 변수의 값에 값을 곱합니다.
/=	기존 변수의 값에 값을 나눕니다.
%=	기존 변수의 값에 나머지를 구합니다.

```
<head>
  <meta charset="utf-8" />

  <title>JavaScript Test</title>
  <script>
        var variable1 = 10;

        variable1 += 30;
        alert(variable1);
  </script>
</head>
```

1.4 변수와 연산자

- 증감 연산자
  - 복합 대입 연산자를 간략하게 사용한 형태

연산자	설명
변수++	기존의 변수의 값에 1을 더합니다(후위).
++변수	기존의 변수의 값에 1을 더합니다(전위).
변수--	기존의 변수의 값에 1을 뺍니다(후위).
--변수	기존의 변수의 값에 1을 뺍니다(전위).

1.4 변수와 연산자

- 변수의 재 선언


```
<script>
    // 변수를 선언합니다.
    var favoriteFood = '김치 찌개';
    var favoriteFood = '라면';
    var favoriteFood = '냉면';
     // 출력합니다.
     alert(favoriteFood);
</script>
```

- 기존에 제공된
  식별자를 재 선언하면
  문제 발생의 여지가 있다

```
<script>
    // 변수를 선언합니다.
    var alert = 'Red Alert';

    // 출력합니다.
    alert(alert);
</script>
```

1.4 변수와 연산자

- typeof 연산자
  - 자료형을 확인할 때 사용


```
alert(typeof "Hello");
alert(typeof 100);
alert(typeof false);
alert(typeof function() { });
alert(typeof { });
alert(typeof tmpvar);
```

1.4 변수와 연산자

- undefined 자료형이란?
  - '존재하지 않는 것'을 표현하는 자료형
  - 변수를 선언했지만 초기화하지 않았을 때


```
<script>

  var myArray;

  alert(myArray);

</script>
```

1.4 변수와 연산자

- 강제로 자료형 변환시키기
  - 다른 자료형을 숫자로 - Number() 함수
  - 다른 자료형은 문자열로 - String() 함수


```
<script>

  var myNum="100";

  alert(Number(myNum) + 300);

</script>
```

```
<script>

  var myString=true;

  alert(String(myString) + "World");

</script>
```

1.4 변수와 연산자

- 일치 연산자의 용도
  - 자료형이 다른 것을 확실하게 구분 짓고 싶을 때 사용

연산자	설명
===	양 변의 자료형과 값이 일치합니다.
!==	양 변의 자료형과 값이 다릅니다.

```
var var1 = 100;
var var2 = "100";

alert("두 변수의 값만을 비교 : " + (var1 == var2) );

alert("두 변수의 Type과 값을 비교 : " + (var1 === var2) );
```

1.5 조건과 반복

- if 조건문
  - 자바스크립트에서 가장 일반적인 조건문 형태
  - 불리언 표현식이 true면 문장 실행
  - false면 문장 무시
  - 조건문에 의해 여러 문장을 실행할 때는 중괄호로 감싸야 함


```
<script>

  var myString="Hello";

  if( myString == "Hello" ) {
      alert("true입니다.");
  } else {
      alert("false입니다.");
  }

</script>
```

1.5 조건과 반복

- if 조건문
  - 변수가 undefined 인지를 확인하는 코드


```
<script>

    var myName;

    if( typeof myName == "undefined" ) {
        alert("myName은 undefined입니다.!!");
    } else {
        alert("myName은 undefined가 아닙니다.!!");
    }

</script>
```

1.5 조건과 반복

- 현재 시간이 오전인지 오후인지를 출력하는 프로그램
  - Date 객체 이용
  - if문 이용


```
<script>

  var date = new Date();
  var hour = date.getHours();

  if( hour < 12 ) {
      alert("오전 입니다.");
  } else {
      alert("오후 입니다.");
  }

</script>
```

1.5 조건과 반복

- Date 객체

-예를 들어 2010-01-14를 세팅하고 싶다면 아래처럼 사용할 수 있다.

- var myDate=new Date();
  myDate.setFullYear(2010,0,14);

getDate()	일자 반환 (from 1-31)
getDay()	요일 반환 (from 0-6)
getFullYear()	4자리 년도 반환 (four digits)
getHours()	시간 반환 (from 0-23)
getMilliseconds()	밀리세컨드 반환 (from 0-999)
getMinutes()	분 반환 (from 0-59)
getMonth()	월 반환 (from 0-11)
getSeconds()	초 반환 (from 0-59)

1.5 조건과 반복

- switch 조건문의 기본 형태
  - default 부분은 생략 가능


```
var var1 = 100;

switch(var1) {
    case 50  : alert("50입니다.");
               break;
    case 100 : alert("100입니다.");
               break;
    case 150 : alert("150입니다.");
               break;
    case "100" : alert("문자 100입니다.");
                 break;
    default : alert("조건에 걸리는 것이 없습니다.");
}
```

1.5 조건과 반복

- 반복문
  - 여러 번 반복해야 하는 일을 간편하게 처리하는 구문

- while 반복문
  - 조건이 변화하지 않는다면 반복문 안을 무한히 반복
  - 조건을 거짓으로 만들 수 있는 내용이 문장 안에 포함돼야 함

```
<script>
var flag = true;
var count = 0;
while(flag){
count++;
if(count==5)
flag = flase;
alert("count의 값은 " + count);
}

//alert((30>tmp)&&(tmp<10));
</script>
```

1.5 조건과 반복

- for 반복문
  - while 반복문은 조건에 비중을 두는 반복문
  - 조건보다 횟수에 비중을 둘 때 for 반복문 사용
  - while 반복문과 달리 초기식과 종결식 있음


```
<script>
   var count = 0;
   for(i=0; i<5; i++) {
         count++;
         alert("count의 값은 " + count);
   }
</script>
```

1.5 조건과 반복

- 브라우저의 성능을 측정하는 프로그램
  - 1초 동안 반복문이 몇 회 반복되는지 표시하는 프로그램
  - 반복문은 1000밀리초(=1초) 후 종료


```
<script>
   var startTime = (new Date()).getTime();
   var count = 0;
   while((new Date()).getTime() < startTime+1000) {
        count++;
   }
   alert("count의 값은 : " + count);
</script>
```

1.5 조건과 반복

- 자바스크립트의 for in 반복문
  - 배열이나 객체를 쉽게 다룰 수 있게 제공하는 반복문

```
<script>
  var arr = ["딸기","바나나","오렌지"];
  for(var i in arr) {
      alert(arr[i]);
  }
</script>
```


```
<script>
  // 배열 선언
  var array = ['포도', '사과', '딸기',
               '바나나'];
  // 반복문
  for(var i in array){
    alert(array[i]);
  }
</script>
```

```
<script>
  // 배열 선언
  var array = ['포도', '사과', '딸기',
               '바나나'];
  // 반복문
  for(var i = 0; i < array.length; i++){
    alert(array[i]);
  }
</script>
```

1.5 조건과 반복

- break 키워드
  - switch 조건문이나 반복문을 벗어날 때 사용하는 키워드
  - 인접한 가장 가까운 loop만을 벗어난다.

- continue 키워드
  - 반복문 내에서 반복을 멈추고 다음 반복을 진행시키는 키워드

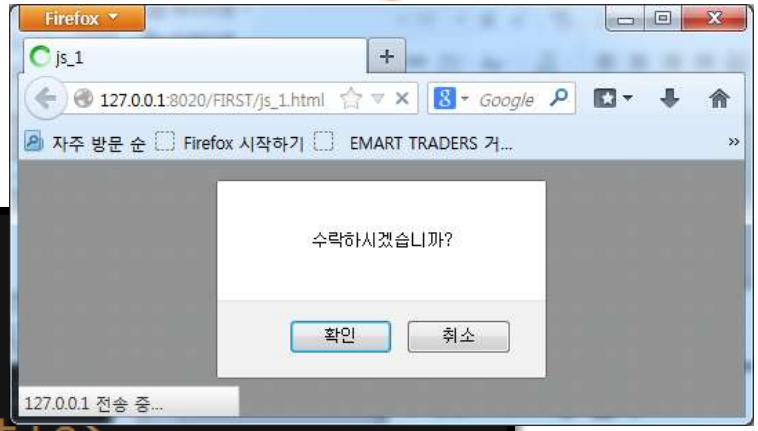
```
var var1 = 100;

for( var i = 0; i < var1; i++ ) {
    continue;
    alert("i의 값은 : " + i);
}

alert("출력이 종료되었습니다.!!");
```

1.6 함수

- 함수란?
  - ✓ 특정 작업을 하는 코드의 집합
  - ✓ 선언적 함수와 익명함수 2가지 종류가 있다.
  - ✓ 일반적으로 입력값과 return값이 존재한다.

JavaScript

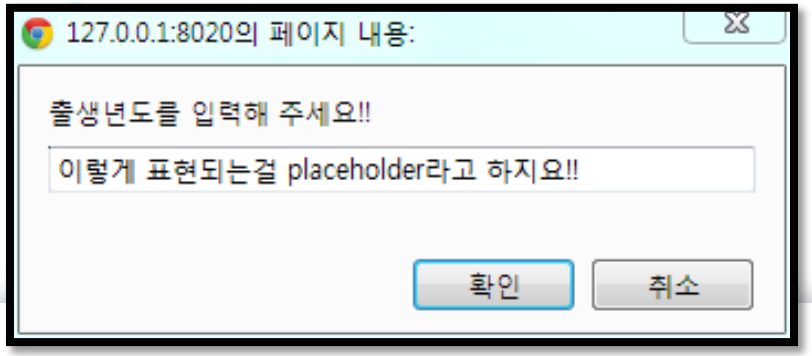
- confirm() 함수
  - ✓ 사용자가 확인을 누르면 true 리턴
  - ✓ 취소를 누르면 false 리턴




```
<head>
  <meta charset="utf-8" />

  <title>JavaScript Test</title>
  <script>

    var result = confirm("수락하시겠습니까?");

  </script>
</head>
```

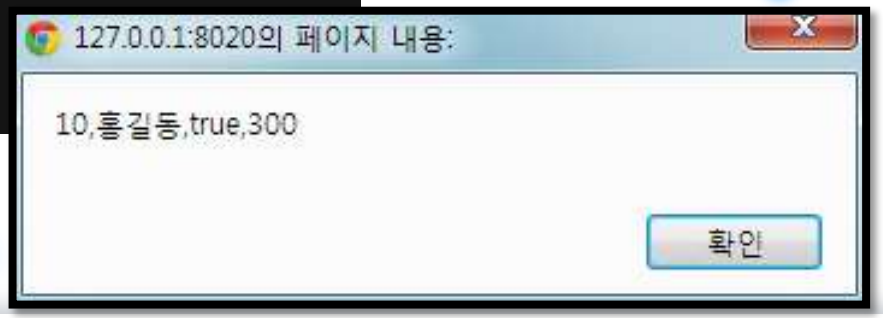
- 숫자를 입력 받는 방법
  - ✓문자열을 입력 받은 후 숫자로 변환
  - ✓문자열을 입력을 할 때 사용하는 함수는 prompt()

```
var title = "출생년도를 입력해 주세요!!";
var content = "이렇게 표현되는걸 placeholder라고 하지요!!";

var result = prompt(title,content);
var age = (new Date()).getFullYear() - parseInt(result) + 1;

alert("당신의 나이는 : " + age);
```



- 배열이란?
  - ✓ 여러 개의 변수를 한꺼번에 다룰 수 있는 자료
  - ✓ 모든 형태의 변수를 다룰 수 있는 자료형
  - ✓ 객체 중 하나
  - ✓ 대괄호([ ])를 사용해 생성
  - ✓ 안에 쉼표로 구분해 자료 입력 (배열 요소라 부름)

```
<script>

  var myArray = [10,"홍길동",true,300];

  alert(myArray);
</script>
```



- 배열 내용 출력 예제
  - ✓ 배열의 맨 앞에 있는 요소는 0번째
  - ✓ 인덱스 – '몇 번째' 라 불리는 숫자

```
<script>

  var myArray = [10,"홍길동",true,300];

  alert(myArray[0]);
  alert(myArray[1]);
  alert(myArray[2]);
  alert(myArray[3]);
</script>
```

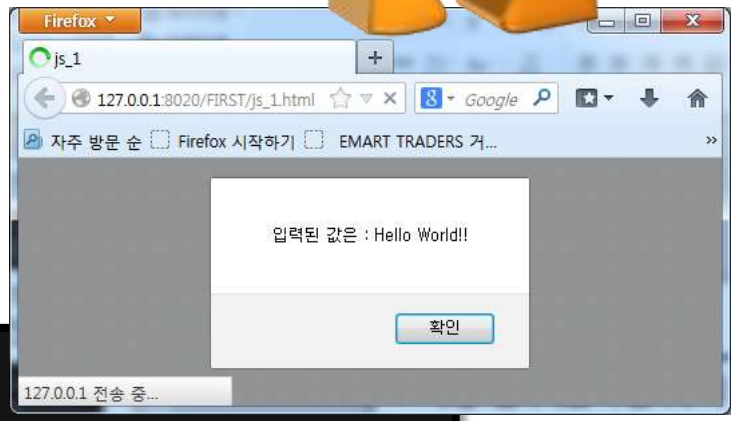
- 익명 함수(람다함수)
  - ✓ 함수도 객체
  - ✓ 함수도 생성자를 사용해서 생성 가능
  - ✓ 함수를 변수에 배정하는 것도 가능

```
sayHi = new Function("toWhom","alert('Hi ' + toWhom);");
sayHi("World");
```

  - ✓ 이런 종류의 함수를 익명함수라 함
  - ✓ 직접적으로 선언하거나 명명한 적이 없기 때문
  - ✓ 익명함수는 동적으로 생성됨

JavaScript

- 익명 함수(람다함수)의 형태
  - ✓ var testFunc = function ( ) { };
  - ✓ 괄호 내부에 코드를 넣음




```
<script>
  // 익명함수의 사용
  var testFunc = function(input) {
      alert("입력된 값은 : " + input);
  }

  testFunc("Hello World!!");
</script>
```

- 익명 함수의 형태

✓ 변수같지만 alert으로 확인해 보면 함수인 것을 확인할 수 있다.

✓ 함수의 이름이 없으므로 변수에 저장해서 사용해야 한다.

```
var myFunc = function() {
    alert("익명함수입니다.!!");
}

alert(function() {
    alert("어떻게 출력될까요??");
});


alert(myFunc);

alert(myFunc());

alert(alert("test"));
```

JavaScript

- 내장 함수의 출력
  - ✓ 내장함수: JavaScript에서 기본적으로 제공하는 함수
  - ✓ 모든 브라우저는 내장하고 있는 함수의 소스를 볼 수 없게 막아놓음.

```
alert(alert);

alert(prompt);
```

- 선언적 함수의 형태
  - ✓ function 함수이름( ) { }
  - ✓ 기존의 프로그래밍에서 함수처럼 사용
  - ✓ 익명 함수와 마찬가지 방법으로 만들고 사용

```
<script>
  function testFunc(input) {
      alert("입력된 값은 : " + input);
  }

  testFunc("Hello World!!");
</script>
```

- 함수사용의 여러 가지 형태
  - 출력 결과는?
  - JavaScript가 실행되다 함수를 호출하는 부분을 만나면 현재의 실행을 잠시 중단하고 제어를 매개변수와 함께 호출한 함수로 넘긴다.

```
function writeOther() {
document.write(3);
}
document.write(1);
writeOther();
document.write(2);
```

JavaScript

- 매개변수 처리
  - 매개변수 처리에 관해 JavaScript는 상당히 유연하다.
  - 함수를 호출할 때 넘기는 인수의 개수와 함수에 선언된 매개변수의 개수가 일치하지 않아도 오류가 발생하지 않는다.
  - 인수가 더 많을 경우 매개변수 수 보다 초과하는 인수는 무시한다.
  - 인수가 매개변수 수 보다 적은 경우 남는 매개변수에 undefined를 할당한다.
  - 또한 인수에 대한 타입체크가 없으며, 어떠한 값이 넘어와도 그대로 매개변수에 할당한다.

```
<script>
function a() {
var b = ""
for(var i=0 ; i < arguments.length ; i++ ) b = b +
arguments[i] + ",";
alert(b);
}
a("123","asdf", 35323, "4343");
</script>
```

- 함수사용의 여러 가지 형태
  - 함수는 일반적으로 아래와 같은 형태로 사용

```
<script>

  function testFunc(input) {
      return input * 100;
  }

  var result = testFunc(10);
  alert("함수호출의 결과값은 : " + result);

</script>
```

JavaScript

- 매개 변수란?
  - ✓ 함수를 호출하는 쪽과 호출된 함수를 연결하는 매개가 되는 변수
  - ✓ 자바스크립트는 함수를 생성할 때 지정한 매개 변수보다 많거나 적은 매개 변수를 사용하는 것 허용
  - ✓ 원래 함수에서 선언된 매개 변수보다 많게 사용하면 추가된 매개 변

```
function argTest(input1, input2) {
    alert(input1);
    alert(input2);
}

argTest();
argTest(1);
argTest(1,2);
argTest(1,2,3);
```

- Array() 함수
  - ✓ 지정한 매개 변수보다 많거나 적게 매개 변수를 사용하는 데 유용


```
<script>
   var arr1 = new Array();
   var arr2 = new Array(5);
   var arr3 = new Array(9,8,7,6,5);

   alert("arr1의 값은 " + arr1 + "\n" +
         "arr2의 값은 " + arr2 + "\n" +
         "arr3의 값은 " + arr3);

</script>
```

- Array() 함수를 사용하지 않고 인자로 들어온 모든 수를 더해서 결과를 alert()으로 출력하려면?


```
<script>
  function sumAll() {
        var sum = 0;
        for(var i=0; i<arguments.length; i++)
            sum += arguments[i];
        return sum;
  }

  var result = sumAll(5,6,7,8,9);
  alert("함수호출의 결과는 : " + result);

</script>
```

- 내부 함수
  - ✓ 함수 내부에 선언한 함수
  - ✓ 내부 함수는 내부 함수를 포함하는 함수 내부에서만 사용가능
  - ✓ 내부 함수를 사용하면 외부에 이름이 같은 함수가 있어도 내부 함수 우선

```
function parentFunc(input1) {

    function childFunc1(input2,input3) {
        alert("여기는 child1 입니다.");
    };

    function childFunc2(input2,input3) {
        alert("여기는 child2 입니다.");
    };

    childFunc1();
}

parentFunc();
childFunc1();  // Exception
```

JavaScript

- 아래의 코드에 오류가 있어 웹 페이지 오류가 발생하여 경고창 미출력 됨.
- 함수 안에 있는 변수는 지역 변수이므로 외부에서 사용할 수 없음.

```
function parentFunc(input1) {

    var myName = "Hello" + input1;

}

alert(myName);
```

JavaScript

- 지역 변수 output은 함수 outerFunction을 실행할 때 생성
  - ✓ 지역 변수이므로 함수가 종료됨과 동시에 사라져야 정상
  - ✓ 코드는 에러를 내야 하나 정상적으로 실행.
  - ✓ 자바스크립트 스스로 아직 지역 변수 output을 지우면 안 된다는 것을 인식하고 남겨두므로 발생하는 특성. (클로져 - 지역 변

```
function outerFunc(input1) {

    var output = input1;
    return function(input2) {
        alert(output + " ---- " + input2);
    }

}

outerFunc("첫번째 인자예요!!")("두번째 인자예요!!");
```

● 클로져

– 클로저(closure)는 자유로운(구속되지 않은) 변수들을 포함하고 있는 코드 블록이다. 이러한 변수들은 코드 블록이나 글로벌 콘텍스트에서 정의되지 않고, 코드 블록이 정의된 환경에서 정의된다. "클로저"라는 명칭은 실행할 코드 블록(자유 변수의 관점에서, 변수 레퍼런스와 관련하여 폐쇄적이지 않은)과 자유 변수들에 대한 바인딩을 제공하는 평가 환경(범위)의 결합에서 탄생한 것이다. 클로저 지원의 다양함은 Scheme, Common Lisp, Smalltalk, Groovy, JavaScript, Ruby, Python에서 찾아볼 수 있다. 클로저의 가치는 함수 객체(*function objects*) 또는 익명 함수(*anonymous functions*)로서 작용하고, 유형 시스템(type system)이 데이터뿐만 아니라 코드도 나타낼 수 있어야 한다는 점에서 유형 시스템에 대한 결과도 갖고 있다. 클로저가 있는 대부분의 언어들은 함수들을 퍼스트-클래스 객체들로서 지원하는데, 함수들은 변수에 저장될 수 있고, 매개변수로서 다른 함수들에 저장되며, 동적으로 생성되고, 함수들에서 리턴된다.

– - IBM developer works 인용 -

- 클로져
  - 어떤 함수를 감싸는 외부 함수가 종료 되었더라도, 내부 함수에서
    외부 함수의 로컬 변수에 접근할 수 있는 방법

- 클로져를 사용할 때 발생할 수 있는 문제
  - 클로저는 부모 함수 안에 있는 변수를 참조
  - 클로저가 사용하는 변수 값이 기대하는 값이 아닐 수도 있음
  - 클로저가 사용하는 변수 값이 반복자의 마지막 값을 참조하게 될
    경우 문제가 발생하기 쉬움

JavaScript

- 클로져의 예
  - ✓ 아래 코드를 살펴보자.
  - ✓ example 함수가 호출될 때마다, i가 초기화 되므로 매번 1이 출력
    될 것 같지만 그렇지 않다.
  - ✓ javascript 는 클로저를 지원하는 언어이기 때문에 return function(){} 이 선언되는 시점의 환경, 즉 지역변수 i의 레퍼런스를 가지고 있기 때문에 계속해서 i를 증가 시킬 수 있다.

```
<script type="text/javascript">
var example = function(){
    var i = 1;
    return function(){
        alert(i++);
    };
}();
</script>

<input type="button" onclick="example()" value="click" />
```

- 함수를 리턴하는 함수(클로져)
  - 특정한 함수에서 자신을 감싸고 있는 함수의 환경(변수나 중첩함수 등)을 그대로 활용할 수 있게 만들어지는 함수

```
<script>
var sequencer = function() {
    var s = 0;
    return function() {
        return ++s;
    }
};

var seq = sequencer();

alert(seq()); // 1
alert(seq()); // 2
alert(seq()); // 3
</script>
```

- JavaScript 기본 내장 함수

함수 이름	설명
eval(string)	string을 자바스크립트 코드로 실행합니다.
isFinite(number)	number가 무한한 값인지 확인합니다.
isNaN(number)	number가 NaN인지 확인합니다.
parseInt(string)	string을 정수로 바꿉니다.
parseFloat(string)	string을 유리수로 바꿉니다.

```
<script>
  var tmp = "alert('이렇게 수행되요!!')";

  eval(tmp);

</script>
```

- JavaScript 기본 내장 함수

```
<script>
  var tmp = 100;
  var result = isNaN(tmp);
  alert(result);
  tmp = "문자열";
  alert(isNaN(tmp));
</script>
```

```
<script>
  var tmp = "100";
  var result = parseInt(tmp);
  result += 100;
  alert("덧셈의 결과는 : " + result);
</script>
```

JavaScript

- JavaScript 객체 생성과 사용 (배열과 비교해서 이해)

```
<script>
   var product = {
        name : "YF Sonata",
        price : 1000,
        color : "white",
        cc : 2000
   }
   alert("제품의 이름은 : " + product.name);
   alert("제품의 색상은 : " + product.color);
</script>
```

- 객체의 키
  - ✓ 대부분 개발자가 식별자를 키로 사용
  - ✓ 식별자로 사용할 수 없는 단어를 키로 사용할 때는 문자열 사용 – 이 경우 대괄호 사용해야 객체 요소 접근 가능

```
var myObj = {
    myName : "홍길동",
    myAge : 30,
    "my Address" : "서울시 성북구",
    "my !!##hobby##!!" : "LOL"
}

alert(myObj.myName);
alert(myObj["my Address"]);
alert(myObj["my !!##hobby##!!"]);
```

- 요소, 속성, 메소드
  - ✓ 배열 내부에 있는 값 – 요소 (element)
  - ✓ 객체 내부에 있는 값 – 속성 (property)
  - ✓ 메소드 (method) - 객체의 속성 중 함수 자료형
  - ✓ this keyword – this keyword가 사용되는 시점에 현재 사용되고 있는 객체를 지칭하는 reference

- JavaScript 객체 생성과 사용 (배열과 비교해서 이해)

```
<script>
  var product = {
        name : "YF Sonata",
        color : ["white","black","red"],
        myfunc : function () {
            return "객체의 method랍니다.!!";
        }
  }
  alert("제품의 이름은 : " + product.name);
  alert("제품의 색상 배열은 : " + product.color);
  alert("제품의 두번째 색상은 : " + product.color[1]);
  alert("제품의 method호출 : " + product.myfunc());
</script>
```

JavaScript

- JavaScript 객체 생성과 사용 (배열과 비교해서 이해)

```
<script>
  var product = {
      name : "YF Sonata",
      color : ["white","black","red"],
      myfunc : function () {
          return "객체의 method랍니다.!!";
      }
  }
  var output ="";
  for( var k in product ) {
      output += product[k];
      output += "\n";
  }
  alert(output);
</script>
```

- 객체 관련 키워드
  - ✓ in 키워드 : 해당 키가 객체 안에 있는지 확인

```
var myObj = {
    myName : "홍길동",
    myAge : 30,
    "my Address" : "서울시 성북구",
    "my !!##hobby##!!" : "LOL"
}

if( "myAge" in myObj ) {
    alert("myAge라는 key가 존재합니다.");
}

if( "my !!##hobby##!!" in myObj ) {
    alert("my Address라는 key가 존재합니다.");
}
```

- 객체 관련 키워드
  - ✓ with 키워드 - 복잡하게 사용해야 하는 코드를 짧게 줄여주는 키워드
  - ✓ 아래 예제에서 myObj를 생략할 수 있음

```
var myObj = {
    myName : "홍길동",
    myAge : 30,
    "my Address" : "서울시 성북구",
    "my !!##hobby##!!" : "LOL"
}

with(myObj) {
    alert("이름 : " + myName);
    alert("나이 : " + myAge);
    alert("취미 : " + myObj["my !!##hobby##!!"]);
}
```

- 동적인 객체 생성과 추가
  - ✓ 처음 객체를 생성하는 시점 이후에 객체의 속성을 추가하거나 제거

```
<script>
   var product = {}
   product.name = "제네시스";
   product.cc = 3000;
   product.remainFuel = 500;
   product.fuel = function(input) {
       this.remainFuel += input;
   };

   alert(product.name);
   alert(product.remainFuel);
   product.fuel(300);
   alert(product.remainFuel);
</script>
```

- 동적인 객체 속성 제거 - delete 키워드 사용

```
<script>
  var product = {}
  product.name = "제네시스";
  product.cc = 3000;
  product.remainFuel = 500;
  product.fuel = function(input) {
      this.remainFuel += input;
  };
  alert(product.name);
  delete(product.name);
  alert(product.name);
</script>
```

- 배열의 push() 메서드 사용 - 배열에 요소를 집어 넣음

```
<script>
  var student = [];
  student.push({ name : "홍길동", kor : 30, eng : 20, math : 80 });
  student.push({ name : "최길동", kor : 65, eng : 46, math : 20 });
  student.push({ name : "박길동", kor : 21, eng : 87, math : 67 });
  student.push({ name : "김길동", kor : 99, eng : 95, math : 100 });
  student.push({ name : "이길동", kor : 80, eng : 33, math : 10 });
</script>
```

- 위의 내용을 가지고 전체 평균을 계산해서 alert()을 이용하여 출력해보자!!

JavaScript

```
<script type="text/javascript">
var students = [];
students.push({name:"하하", kor : 30, eng: 20});
students.push({name:"하하2", kor : 40, eng: 30});
var Avg = function(){
    this.sum = 0;
    this.len = 0;
    this.addPoint = function(Point)
    {   this.sum += Point;
        this.len += 1;   }
    this.getAvg = function()
    {     return this.sum / this.len;   }
}

var korAvg  = new Avg();
var mathAvg = new Avg();
var engAvg  = new Avg();

var output = '이름\t총점\t평균\t등급\n';
for (var i in students)
{
    output += students[i].name + '\n';
    korAvg.addPoint(students[i].kor);
    engAvg.addPoint(students[i].eng);
}
output += '평  균\t국어평균:' + korAvg.getAvg() + '\t영어평균:' + engAvg.getAvg();
alert(output);
</script>
```

- 객체를 개별적으로 생성할 때의 이점
  - ✓ 서로 다른 형태의 객체를 배열 안에 넣을 수 있다는 장점
- 같은 종류의 객체를 데이터만 다르게 해서 만들 시에는 불필요한 코드 작업 필요.
- 같은 종류의 객체를 데이터만 다르게 해서 만들 시에는
  - ✓ 객체 생성 함수를 따로 작성
  - ✓ 생성자 함수 이용

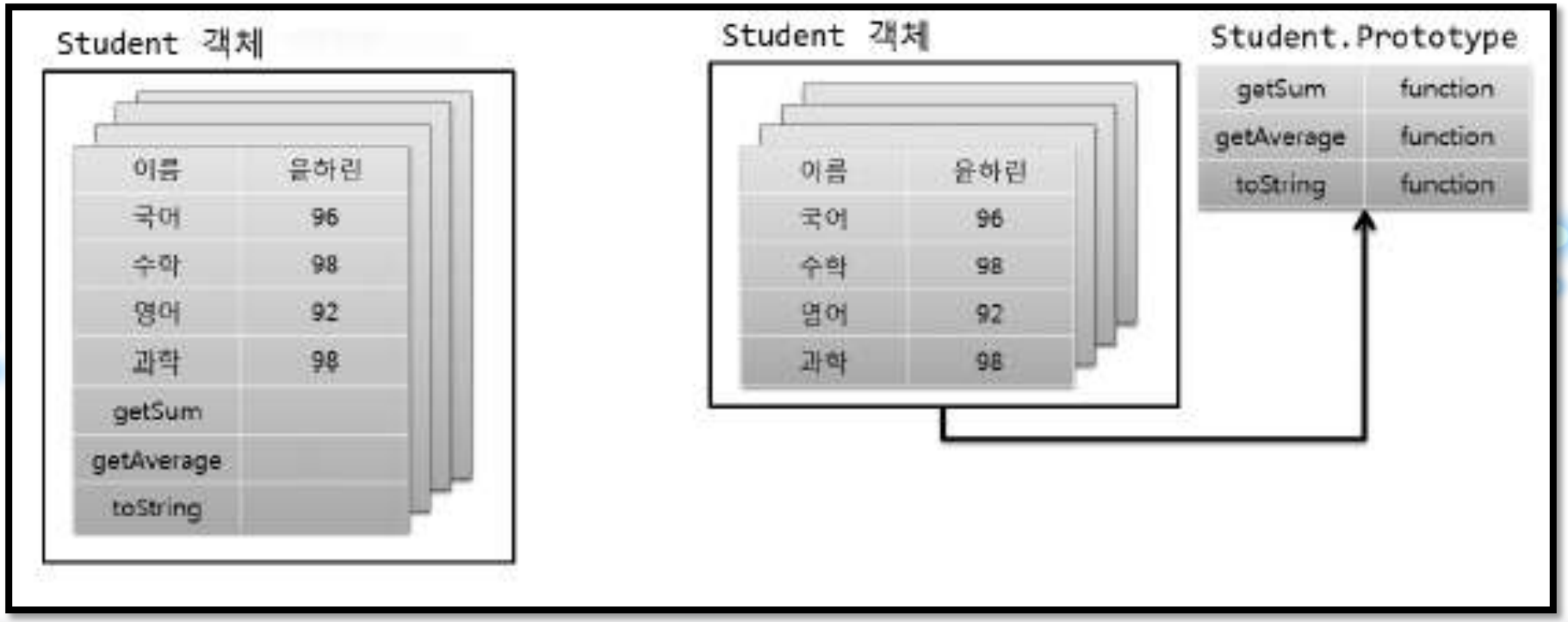
- 객체 생성 함수 – 빠르고 쉽게 객체를 생성하는 방법

```
<script>
  function makeObj(name,kor,eng,math) {
      var student = {
          sName : name,
          sKor : kor,
          sEng : eng,
          sMath : math,

          getAverage : function () {
              var tot = kor+eng+math;
              return tot/3.0;
          }
      };
      return student;
  }
  alert("평균은 : " + makeObj("홍길동",40,90,80).getAverage());
</script>
```

- 생성자 함수
  - ✓new 키워드를 사용해 객체를 생성할 수 있는 함수
  - ✓this 키워드 사용해 생성될 객체의 속성 지정

```
<script>
  function Student(name,kor,eng,math) {
        this.name = name;
        this.kor = kor;
        this.eng = eng;
        this.math = math;

        this.getAverage = function () {
            var tot = kor + eng + math;
            return tot/3.0;
        }

  }
  var s1 = new Student("홍길동",40,90,80);
  var s2 = new Student("김길동",90,90,60);
  alert("홍길동의 평균은 : " + s1.getAverage());
  alert("김길동의 평균은 : " + s2.getAverage());
</script>
```

- 객체 생성 함수와 생성자 함수의 비교
  - ✓ 속성은 모든 객체가 다른 값을 가지지만 메서드는 모두 같은 값
  - ✓ 각기 객체를 생성할 때마다 동일한 함수를 계속 생성하는 것은 낭비



JavaScript

```
var myObj1 = {
    myName : "홍길동",
    myAge : 30,
    "my Address" : "서울시 성북구",
    "my !!##hobby##!!" : "LOL",
    printMyInfo : function() {
        alert(this.myName + ", " + this.myAge);
    }
}

var myObj2 = {
    myName : "김길동",
    myAge : 35,
    "my Address" : "인천",
    "my !!##hobby##!!" : "World of WarCraft",
    printMyInfo : function() {
        alert(this.myName + ", " + this.myAge);
    }
}

alert(myObj1.prototype);
```

JavaScript

```
function MyObj(name, age, address, hobby) {
    this.name = name;
    this.age = age;
    this.address = address;
    this.hobby = hobby;
    this.printMyInfo = function() {
        alert(name + ", " + age);
    }
}

obj1 = new MyObj("홍길동",30,"서울","LOL");
obj2 = new MyObj("김길동",40,"인천","World of WarCraft");

alert(MyObj.prototype);
```

- 생성자 함수 사용
  - ✓ 내부에는 속성만 존재
  - ✓ 메서드는 프로토 타입에 들어간다.
  - ✓ 프로토 타입 사용한 메서드 생성

```
<script>
    function Student(name, korean, math, english, science) {
        this.이름 = name;
        this.국어 = korean;
        this.수학 = math;
        this.영어 = english;
        this.과학 = science;
    }

    Student.prototype.getSum = function () { };
    Student.prototype.getAverage = function () { };
    Student.prototype.toString = function () { };
</script>
```

- instanceof 키워드
  - ✓ 생성자 함수를 통해 만들어진 객체가 인스턴스(instance)
  - ✓ 해당 객체가 어떠한 생성자 함수를 통해 생성됐는지 확인할 때 사용

```
function MyObj(name, age, address, hobby) {
    this.name = name;
    this.age = age;
    this.address = address;
    this.hobby = hobby;
    this.printMyInfo = function() {
        alert(name + ", " + age);
    }
}

obj1 = new MyObj("홍길동",30,"서울","LOL");
obj2 = new MyObj("김길동",40,"인천","World of WarCraft");

alert(obj1 instanceof MyObj);
// true or false
```

- instanceof 와 tyepof
  - ✓ typeof는 unary 오퍼레이터이다. unary 오퍼레이터로는 ! 라던가 - 등과 같이 인자를 하나만 받을 수 있는 연산자를 뜻한다. 즉, 함수가 아니고 연산자이기 때문에 괄호를 사용하면 안된다.

```
if(typeof yourVariable === 'object') { /* 오브젝트 처리 */}
```

  - ✓ instanceof 는 비교 연산자로 >,<,== 와 같이 두개의 인자를 받는 연산자로 앞의 비교 연산자들을 이용하는 기분으로 사용하면 된다. 하지만 결과로 리턴하는 것은 typeof와는 성질이 조금 다르다. instanceof는 해당하는 변수가 사용하고 있는 prototype의 chain을 2번째 인자와 쭉 비교해서 true/false 값을 리턴한다.

- 기본 자료형이란?
  - ✓ 숫자, 문자열, 불리언
- 기본 자료형과 객체의 차이
  - ✓ 속성과 메소드는 객체만 가질 수 있음
  - ✓ 기본 자료형의 속성이나 메소드 사용시 - 기본 자료형이 자동 객체 변환
- 기본 자료형에 메소드를 추가해서 사용할 수 있는가

```
var primitiveNumber = 100;

primitiveNumber.myFunc = function() {
    alert("함수추가");
}

primitiveNumber.myFunc();
```

- 기본 자료형에 메소드를 추가할 수 있다.
  - ✓ 함수의 프로토타입에 메소드 추가

```
var primitiveNumber = 100;

Number.prototype.myFunc = function() {
    alert("함수추가");
}

primitiveNumber.myFunc();
```

- JavaScript 내장객체 - Number 객체
  - ✓ 자바스크립트에서 가장 단순한 객체

```
<script>

   var tmp = 3.1415926535;
   var tmpObj = new Number(3.1415926535);

   alert(tmp);
   alert(tmpObj);
   alert(tmp.toFixed(3));
</script>
```

```
var n = 123456.789;

n.toFixed(0);   //"1234567"
n.toFixed(2);   //"123456.79"

n.toExponential(1);   //"1.2e+57"
n.toExponential(3);   //"1.235+5"

n.toPrecision(4);   //"1.235+5"
n.toPrecision(7);   //"123456.8"
```

- JavaScript 내장객체 - Number 객체
  - ✓Number 생성자 함수의 속성

Property	Description
constructor	생성자
MAX_VALUE	JavaScript로 표현가능한 가장 큰 수
MIN_VALUE	JavaScript로 표현가능한 가장 작은 수
NEGATIVE_INFINITY	음수 무한대를 표현합
NaN	"Not-a-Number" 값,
POSITIVE_INFINITY	양수 무한대
prototype	사용자 속성과 함수 추가시

Method	Description
toExponential(x)	지수 형태로 변환
toFixed(x)	소수점 표현을 x 자릿 수만큼 표현
toPrecision(x)	전체 표현을 x 만큼 표현
toString()	문자열로 변환
valueOf()	Number 객체의 원시 값을 반환

- JavaScript 내장객체 - String 객체
  - ✓ 자바스크립트에서 가장 많이 사용하는 내장 객체

```
<script>

  var testMsg = "This is a sample Text";

  alert(testMsg.length);
  alert(testMsg.charAt(0));
  alert(testMsg.substring(3,6));
  alert(testMsg.toUpperCase());
</script>
```

- JavaScript 내장객체 - String 객체

메서드 이름	설명
charAt(position)	position에 위치하는 문자를 리턴합니다.
charCodeAt(position)	position에 위치하는 문자의 유니코드 번호를 리턴합니다.
concat(string, ... , string)	매개 변수로 입력한 문자열을 이어 리턴합니다.
indexOf(searchString, position)	앞에서부터 일치하는 문자열의 위치를 리턴합니다.
lastIndexOf(searchString, position)	뒤에서부터 일치하는 문자열의 위치를 리턴합니다.
match(regExp)	문자열 내에 regExp가 있는지 확인합니다.
replace(regExp, replacement)	regExp를 replacement로 바꾼 뒤 리턴합니다.
search(regExp)	regExp와 일치하는 문자열의 위치를 리턴합니다.
slice(start, end)	특정 위치의 문자열을 추출해 리턴합니다.
split(separator, limit)	separator로 문자열을 잘라 배열을 리턴합니다.
substr(start, count)	start부터 count만큼 문자열을 잘라서 리턴합니다.
substring(start, end)	start부터 end까지 문자열을 잘라서 리턴합니다.
toLowerCase( )	문자열을 소문자로 바꿔 리턴합니다.
toUpperCase( )	문자열을 대문자로 바꿔 리턴합니다.

- JavaScript 내장객체 - String 객체의 잘못된 사용
  - ✓ toUpperCase는 복사본을 생성함

```
var str = "This is a sample Text";

str.toUpperCase();

alert(str);
```

잘못된 사용

```
var str = "This is a sample Text";

var result = str.toUpperCase()

alert(result);
```

올바른 사용

- String 객체의 HTML 관련 메소드

메서드 이름	설명
anchor( )	a 태그로 문자열을 감싸 리턴합니다.
big( )	big 태그로 문자열을 감싸 리턴합니다.
blink( )	blink 태그로 문자열을 감싸 리턴합니다.
bold( )	b 태그로 문자열을 감싸 리턴합니다.
fixed( )	tt 태그로 문자열을 감싸 리턴합니다.
fontcolor(colorString)	font 태그로 문자열을 감싸고 color 속성을 주어 리턴합니다.
fontsize(fontSize)	font 태그로 문자열을 감싸고 size 속성을 주어 리턴합니다.
italics( )	I 태그로 문자열을 감싸 리턴합니다.
link(linkRef)	a 태그에 href 속성을 지정해 리턴합니다.
small( )	small 태그로 문자열을 감싸 리턴합니다.
strike( )	strike 태그로 문자열을 감싸 리턴합니다.
sub( )	sub 태그로 문자열을 감싸 리턴합니다.
sup( )	sup 태그로 문자열을 감싸 리턴합니다.

- JavaScript 내장객체 - Array 객체의 속성과 메소드

속성 이름	설명
length	배열 요소의 개수를 알아냅니다.

메서드 이름	설명
concat( )	매개 변수로 입력한 배열의 요소를 모두 합쳐 배열을 만들어 리턴합니다.
join( )	배열 안의 모든 요소를 문자열로 만들어 리턴합니다.
pop( )*	배열의 마지막 요소를 제거하고 리턴합니다.
push( )*	배열의 마지막 부분에 새로운 요소를 추가합니다.
reverse( )*	배열의 요소 순서를 뒤집습니다.
slice( )	배열 요소의 지정한 부분을 리턴합니다.
sort( )*	배열의 요소를 정렬하고 리턴합니다.
splice( )*	배열 요소의 지정한 부분을 삭제하고 삭제한 요소를 리턴합니다.

- JavaScript 내장객체 - Array 객체
  - ✓ 앞 페이지 참조

```
<script>

  var testArr = [10,67,34,98,54];

  alert("pop() 의 결과 : " + testArr.pop());
  alert("현재 Array의 값 : " + testArr);
  testArr.push(69);
  alert("push(69) 의 결과 : " + testArr);
  alert("정렬결과 : " + testArr.sort());
</script>
```

JavaScript

- JavaScript 내장객체 - Array 객체 연습문제
- 밑의 Array에서 1등과 2등의 정보(이름, 평균, 등수)를 출력하세요!!.

```
<script>
  var student = [];
  student.push({ name : "홍길동", kor : 30, eng : 20, math : 80 });
  student.push({ name : "최길동", kor : 65, eng : 46, math : 20 });
  student.push({ name : "박길동", kor : 21, eng : 87, math : 67 });
  student.push({ name : "김길동", kor : 99, eng : 95, math : 100 });
  student.push({ name : "이길동", kor : 80, eng : 33, math : 10 });
</script>
```

- JavaScript 내장객체 - Date 객체

```
<script>
    // 변수를 선언합니다.
    var date = new Date();

    // 출력합니다.
    alert(date);
</script>
```

```
var date = new Date('December 9');
var date = new Date('December 9, 1991');
var date = new Date('December 9, 1991 02:24:23');
```

```
var date = new Date(1991, 12, 9);
var date = new Date(1991, 12, 9, 2, 24, 23);
var date = new Date(1991, 12, 9, 2, 24, 23, 1);
```

- JavaScript 내장객체 - Date 객체
  - ✓ to○○String() 형태 메소드

```
var today = new Date(2013,1,1);

var result = "";
result += "toDateString : "
            + today.toDateString() + "\n";
result += "toGMTString : "
            + today.toGMTString() + "\n";
result += "toLocaleDateString : "
            + today.toLocaleDateString() + "\n";
result += "toLocaleString : "
            + today.toLocaleString() + "\n";
result += "toTimeString : "
            + today.toTimeString() + "\n";

alert(result);
```

- JavaScript 내장객체 - Date 객체
  - ✓ to○○String() 형태 메소드

```
var today = new Date("January 1, 2013");

var result = "";
result += "toDateString : "
            + today.toDateString() + "\n";
result += "toGMTString : "
            + today.toGMTString() + "\n";
result += "toLocaleDateString : "
            + today.toLocaleDateString() + "\n";
result += "toLocaleString : "
            + today.toLocaleString() + "\n";
result += "toTimeString : "
            + today.toTimeString() + "\n";

alert(result);
```

- JavaScript 내장객체 - Date 객체
  - ✓ 일주일 후의 날짜 구하기

```
var today = new Date();

today.setDate(today.getDate() + 7);

alert(today.toLocaleString());
```

- JavaScript 내장객체 - Math 객체
  - ✓ 자바스크립트의 기본 내장 객체 중 유일하게 생성자 함수 미사용 객체

속성 이름	값
E	2.718281828459045
LN2	0.6931471805599453
LN10	2.302585092994046
LOG2E	1.4426950408889633
LOG10E	0.4342944819032518
PI	3.141592653589793
SQRT1_2	0.7071067811865476
SQRT2	1.4142135623730951

메서드 이름	설명
abs(x)	x의 절대 값을 구합니다.
acos(x)	x의 아크 코사인 값을 구합니다.
asin(x)	x의 아크 사인 값을 구합니다.
atan(x)	x의 아크 탄젠트 값을 구합니다.
atan2(y, x)	x와 y의 비율로 아크 탄젠트 값을 구해 구합니다.
ceil(x)	x보다 크거나 같은 가장 작은 정수를 구합니다.
cos(x)	x의 코사인 값을 구합니다
exp(x)	자연 로그의 x 제곱을 구합니다.
floor(x)	x보다 작거나 같은 가장 큰 정수를 구합니다.
log(x)	x의 로그 값을 구합니다.
max(x,y,z,…,n)	매개 변수 중 가장 큰 값을 구합니다.
min(x,y,z,…,n)	매개 변수 중 가장 작은 값을 구합니다.
pow(x,y)	x의 y 제곱을 구합니다.
random( )	0부터 1까지의 임의의 수를 구합니다.
round(x)	x를 반올림하여 구합니다.
sin(x)	x의 사인 값을 구합니다.
squt(x)	x의 제곱근을 구합니다.
tan(x)	x의 탄젠트 값을 구합니다.

2.JSON

- JavaScript 내장객체 - JSON (JavaScript Object Notation)
  - ✓ 자바스크립트 객체의 형태를 가지는 문자열을 의미

메서드 이름	설명
JSON.parse( )	JSON 형식의 문자열을 자바스크립트 객체로 만듭니다.
JSON.stringify( )	자바스크립트 객체를 JSON 형식의 문자열로 만듭니다.

JavaScript - JSON

- JavaScript Object Notation
- 인터넷에서 자료를 주고받을 때 그 자료를 표현하는 방법
- 자바스크립트 형식을 따르지만, 프로그래밍 언어나 플랫폼에 독립적이다
- JSON 문법은 자바스크립트 표준인 ECMA-262 3판의 객체 문법
- Unicode 인코딩

JavaScript - JSON

- 표현할 수 있는 자료형 : 숫자, 문자열, true/false, 배열, 객체
- “Key : Value” 구조이며 중괄호로 감싼다.

```
{ name : “Kim” , age : 27 , married : true }
```

- Key 값에 따옴표를 쓸 수 있다 (일반적인 사용법)

```
{ “name” : “Kim” , “age” : 27 , “married” : true }
```

- 순서는 상관 없다

JavaScript - JSON

- 기존 변수선언 방식과의 차이점
  - ✓ 기존 변수 선언 방식

```
var name = "Kim";
var age = 27;
var married = true;
```

  - ✓ JSON

```
{ name : "Kim" , age : 27 , married : true }
```

- 변수선언 방식보다 훨씬 간결하다

JavaScript - JSON

- 기존 변수선언 방식과의 차이점
  - ✓ 기존 변수 선언 방식

```
var car_name = 'SONATA';
var car_color = ['red', 'blue'];
var car_price_red = 100;
var car_price_blue = 200;
```

  - ✓ JSON

```
var car = {name: 'SONATA' , color: { red: 100, blue: 200 } }
```

- 변수선언 방식보다 간결하고, 관계가 잘 나타나 있다

JavaScript - JSON

- 함수의 파라미터로 값을 넘길 때도 간결하다
  - ✓기존 변수 선언 방식

```
var car_name = 'SONATA';
var car_color = ['red', 'blue'];
var car_price_red = 100;
var car_price_blue = 200;

printPrice( car_name, car_color, car_price_red, car_p
rice_blue );
```

  - ✓ JSON

```
var car = {name: 'SONATA' , color: { red: 100, blue:
200 } }

printPrice( car );
```

JavaScript - JSON

- 일반 파라미터 전달 방식과 JSON 방식의 차이를 볼 수 있다.
- JSON을 쓰지 않으면 모든 파라미터를 하나씩 적어서 전달해야 한다.
  - ✓ 파라미터 개수가 몇 개 되지 않으면 문제가 없지만 많아지면 쉽지 않은 일이다.
  - ✓ 개수도 문제이지만 순서도 문제이다.
  - ✓ 순서도 정확히 지켜서 전달 해야 하기 때문이다.

- 반면, JSON 방식은 간편하게 값을 전달 할 수 있는 장점이
  있다.

JavaScript - JSON

- 파라미터 개수가 추가 되어도 함수가 변경되지 않는다
  - ✓기존 변수 선언 방식

```
var car_name = 'SONATA';
var car_color = ['red', 'blue'];
var car_price_red = 100;
var car_price_blue = 200;
var car_type = "Auto";

printPrice( car_name, car_color, car_price_red, car_price_blue, car_
type ); //파라미터 추가 될때마다 printPrice 함수가 변경되어야 한다
```

  - ✓ JSON

```
var car = {name: 'SONATA' , color: { red: 100, blue: 200 }, type:
'Auto' }

printPrice( car );
```

JavaScript - JSON

- JSON의 장점은 파라미터가 추가 될 때 나타난다.
- JSON이 아닌 방법에서는 파라미터가 추가됨에 따라 함수(function) 구조가 바뀌지만 JSON에서는 바뀌지 않는다.
- 잦은 변경에 유연하게 대처 할 수 있는 장점이다.

JavaScript - JSON

- JSON 값의 호출
  - ✓ 일반적인 호출 방법 : “JSON객체”.“Key”

```
var car = {name: 'SONATA' , color: { red: 100, blue: 200 }}

console.log( car.name ); // console.log() 는 콘솔창에 출력합니다.
```

  - ✓ JSON객체안에 JSON객체가 있는 경우 : “JSON객체”.“Key(객체)”.“Key”

```
var car = {name: 'SONATA' , color: { red: 100, blue: 200 }}

console.log( car.color.red );
```

JavaScript - JSON

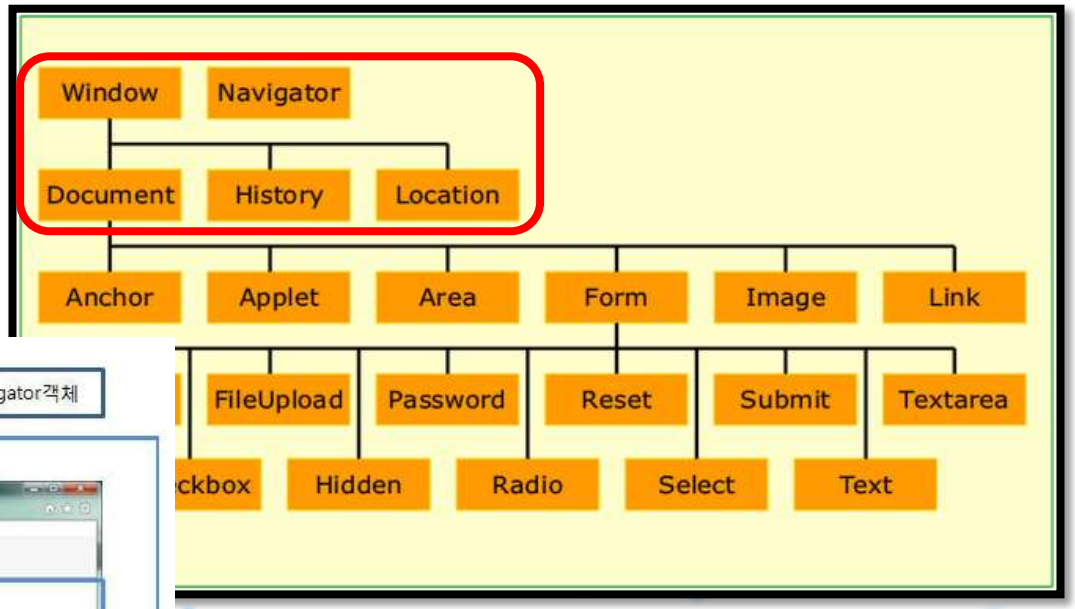
- JavaScript 내장객체 - JSON (JavaScript Object Notation)

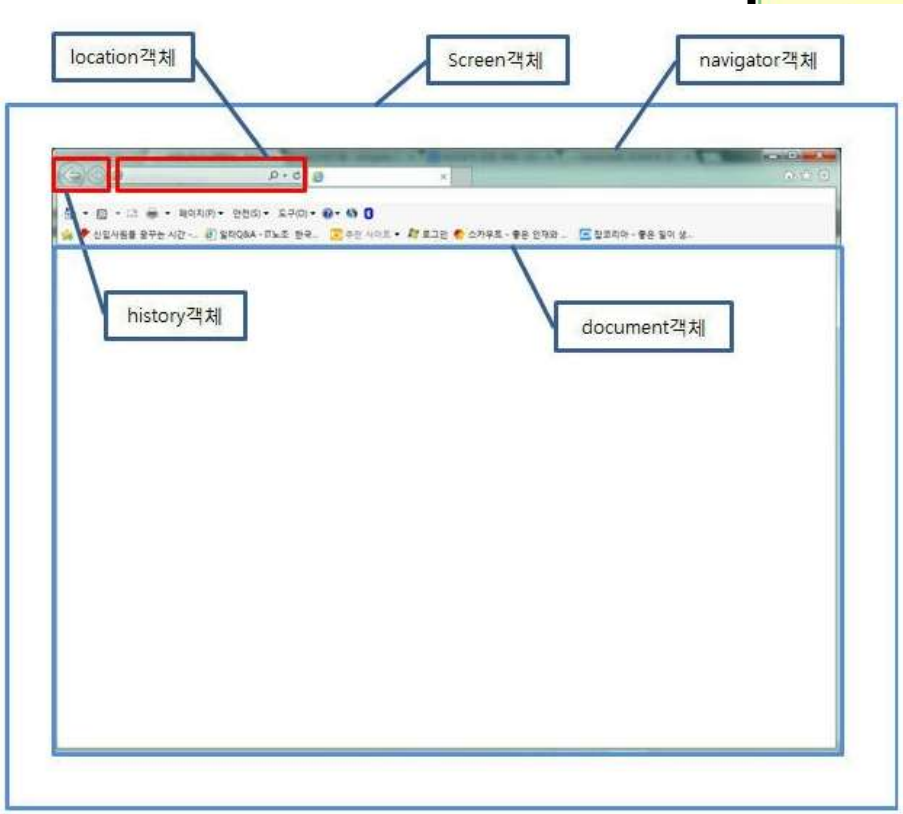
```
<script>
  var testObj = {
      name : "홍길동",
      gender : "남자",
      age : 20
  };

  alert("JSON형태의 객체 표현 : " + JSON.stringify(testObj));

  function makeObj(name,gender,age) {
      this.sName = name;
      this.sGender = gender;
      this.sAge = age;

      this.getAge = function() {
          return this.sAge;
      }
  }
  alert("JSON형태의 객체 표현 : " + JSON.stringify(new makeObj("박길동","남자",30)));
</script>
```

3.DOM

- 브라우저 객체 모델 이란?
  - ✓ BOM, Browser Object Model
  - ✓ BOM(브라우저 객체 모델)은, 웹 브라우저 창을 관리할 목적으로 제공되는 객체 모음을 대상으로 하는 모델로써, 자바 스크립트 등에서 이를 사용할 수 있다.
  - ✓ 브라우저 제작사 마다 세부사항이 다소 다르게 구현, 한정적임
  - ✓ 웹 브라우저와 관련된 객체의 집합
  - ✓ window, location, navigator, history, screen, document 객체

JavaScript


Window
Navigator
Document
History
Location
Anchor
Applet
Area
Form
Image
Link
FileUpload
Password
Reset
Submit
Textarea
ckbox
Hidden
Radio
Select
Text


location객체
Screen객체
navigator객체
history객체
document객체

- window 객체
  - ✓ window 객체의 open() 메소드
  - ✓ 새창을 띄워주는 메소드

메서드 이름	설명
open(URL, name, features, replace)	새로운 window 객체를 생성합니다.

```
<script>
  window.open("http://www.naver.com","child","width=300, height=300",true);
</script>
```

옵션 이름	설명	입력할 수 있는 값
height	새 윈도우의 높이	픽셀 값
width	새 윈도우의 너비	픽셀 값
location	주소 입력창의 유무	yes, no, 1, 0
menubar	메뉴의 유무	yes, no, 1, 0
resizable	화면 크기 조절 가능 여부	yes, no, 1, 0
status	상태 표시줄의 유무	yes, no, 1, 0
toolbar	상태 표시줄의 유무	yes, no, 1, 0

- window 객체
  - ✓ window 객체의 타이머 메소드
  - ✓ 특정한 시간에 특정한 함수를 실행할 수 있게 하는 메소드
  - ✓ setTimeout() 메소드 - 특정한 시간 후에 한 번 함수 실행
  - ✓ setInterval() 메소드 - 특정한 시간마다 함수 실행 (메모리 차지)

메서드 이름	설명
setTimeout(function, millisecond)	일정 시간 후에 함수를 한번 실행합니다.
setInterval(function, millisecond)	일정 시간마다 함수를 반복해서 실행합니다.
clearTimeout(id)	일정 시간 후에 함수를 한번 실행하는 것을 중지합니다.
clearInterval(id)	일정 시간마다 함수를 반복하는 것을 중단합니다.

- window 객체
  - ✓ window 객체의 타이머 메소드

```
<script>
  // 생성한 window 3초후에 종료!!
  var win =
      window.open("http://www.naver.com","child","width=300, height=300",true);

  window.setTimeout(
      function() {
          win.close();
      }
      , 3000
  );
</script>
```

- window 객체
  - ✓ 타이머 멈추기
  - ✓ clearTimeout() 메소드와 clearInterval() 메소드 사용
  - ✓ setTimeout() 메소드와 setInterval() 메소드 사용하면 타이머 아이디 리턴
  - ✓ 타이머 아이디를 매개 변수에 넣어주면 타이머 정지 가능
  - ✓ 10초간 document에 현재시간을 찍어주고 그 이후에는 중지하는 스크립트를 작성해 보자!! (다음 장 참조)

- window 객체
  - ✓ 타이머 멈추기

```
  <script>
    // 10초간 document에 현재시간을 출력하고 그 이후에는 중지!!

    function timer() {
        var startTime = window.setInterval(
            function() {
                document.body.innerHTML = "<b>" + new Date() + "</b>";
            },
            1000
        );

        window.setTimeout(
            function() {
                clearInterval(startTime);
            }
            , 10000
        );
    }
  </script>
</head>
<body onload="timer()">
```

- window 객체
  - ✓기타 method

메서드 이름	설명
moveBy(x, y)	윈도우의 위치를 상대적으로 이동합니다.
moveTo(x, y)	윈도우의 위치를 절대적으로 이동합니다.
resizeBy(x, y)	윈도우의 크기를 상대적으로 지정합니다.
resizeTo(x, y)	윈도우의 크기를 절대적으로 지정합니다.
scrollBy(x, y)	윈도우 스크롤의 위치를 상대적으로 이동합니다.
scrollTo(x, y)	윈도우 스크롤의 위치를 절대적으로 이동합니다.
focus( )	윈도우에 초점을 맞춥니다.
blur( )	윈도우에 초점을 제거합니다.
close( )	윈도우를 닫습니다.

- screen 객체
  - ✓ 웹 브라우저의 화면이 아니라 운영체제 화면의 속성을 가지는 객체
  - ✓ 모든 브라우저가 공통적으로 가지는 screen 객체 속성

속성 이름	설명
width	화면의 너비
height	화면의 높이
availWidth	실제 화면에서 사용 가능한 너비
availHeight	실제 화면에서 사용 가능한 높이
colorDepth	사용 가능한 색상 수
pixelDepth	한 픽셀당 비트 수

```
<script>

  alert("화면 넓이 : " + screen.width);
  alert("화면 넓이 : " + screen.height);
  alert("사용가능한 색상 수 : " + screen.colorDepth);

</script>
```

JavaScript

- location 객체
  - ✓ 브라우저의 주소 표시줄과 관련된 객체
  - ✓ location 객체는 프로토콜의 종류, 호스트 이름, 문서 위치 등의 정보

속성 이름	설명	예
href	문서의 URL 주소	
host	호스트 이름과 포트 번호	localhost:30763
hostname	호스트 이름	localhost
port	포트 번호	30763
pathname	디렉토리 경로	/Projects/Location.htm
hash	앵커 이름(#~)	#beta
search	요청 매개 변수	?param=10
protocol	프로토콜 종류	http:

속성 이름	설명
assign(link)	현재 위치를 이동합니다.
reload( )	새로고침합니다.
replace(link)	현재 위치를 이동합니다.

JavaScript

- location 객체
  - ✓ 브라우저의 주소 표시줄과 관련된 객체
  - ✓ location 객체는 프로토콜의 종류, 호스트 이름, 문서 위치 등의 정보

```
  <script>
     sWidth=screen.width;
     sHeight=screen.height;

     popwin = window.open("http://www.naver.com",
                          "child",
                          "width=" + sWidth + ",height=" + sHeight);

     function change() {
            popwin.location.replace("http://www.daum.net");
     }


  </script>

</head>
<body onload="change()">
```

- navigator 객체
  - ✓ 웹 페이지 실행하고 있는 브라우저에 대한 정보

속성 이름	설명
appCodeName	브라우저의 코드명
appName	브라우저의 이름
appVersion	브라우저의 버전
platform	사용중인 운영체제의 시스템 환경
userAgent	브라우저의 전체적인 정보

```
<script>

    alert(navigator.platform);
    alert(navigator.userAgent);

</script>
```

JavaScript

- window 객체의 onload 속성
  - ✓ 이벤트 속성
  - ✓ window 객체가 로드 완료되고 자동으로 할당한 함수 실행

```
<script>

  window.onload = function() {
      alert("window load!!");
  }

</script>
```

JavaScript

- 문서 객체 모델 (DOM)
  - ✓ 넓은 의미로 웹 브라우저가 HTML 페이지 인식하는 방식
  - ✓ 좁은 의미로는 document 객체와 관련된 객체 집합
  - ✓ 사용시 HTML 페이지에 태그를 추가, 수정, 제거할 수 있음
  - ✓ 문서 객체 - 태그를 자바스크립트에서 이용할 수 있는 객체로 만든 것

- document 객체의 getElementById() 메서드
  - ✓ 문서 객체를 자바스크립트로 가져와 조작 가능

- 문서 객체 모델 (DOM)

```
  <script>

    function domTest() {
        var email = document.getElementById("email");
        email.setAttribute("placeholder","test@abc.com");
    }

  </script>

</head>
<body onload="domTest()">
    <form action="#" method="post">
        <label for="email">Email : </label>
        <input type="email" id="email" name="email" />
    </form>
</body>
```

- 노드
  - ✓ 요소 노드 (Element Node)
  - ✓ 텍스트 노드 (Text Node)

메서드 이름	설명
createElement(tagName)	요소 노드를 생성합니다.
createTextNode(text)	텍스트 노드를 생성합니다.

```
  <script>

    function addElement() {
        var header = document.createElement("h1");
        var txtNode = document.createTextNode("Hello World!!");

        header.appendChild(txtNode);
        document.body.appendChild(header);
    }

  </script>

</head>
<body>
    <form action="#" method="post">
        <input type="button" value="Element 추가!!" onclick="addElement()" />
    </form>
</body>
```

- Text노드를 가지지 않는 경우 (Image)

```
  <script>

    function addElement() {
        var img = document.createElement("img");
        img.src = "img/shin.png"
        img.width=465;
        img.height=346;

        document.body.appendChild(img);
    }

  </script>

</head>
<body>
    <form action="#" method="post">
        <input type="button" value="IMAGE 추가!!" onclick="addElement()" />
    </form>
</body>
```

- 문서 객체의 속성과 관련된 메소드

메서드 이름	설명
setAttribute(name, value)	객체의 속성을 지정합니다.
getAttribute(name)	객체의 속성을 가져옵니다.

JavaScript

```
<head>
  <meta charset="utf-8" />
  <title>index</title>
  <script>
    function createImg() {
        var img = document.createElement("img");
        img.src = "img/shin.png";
        img.width = 465;
        img.height = 346;
        img.setAttribute("id","shinImg");
        document.body.appendChild(img);
    }

    function changeImg() {
        var img = document.getElementById("shinImg");
        img.setAttribute("src","img/boyoung.png")
    }
  </script>
</head>
<body>
    <input type="button" value="이미지 생성" onclick="createImg()" /><br>
    <input type="button" value="이미지 변경" onclick="changeImg()" />
</body>
</html>
```

JavaScript

- 문서 객체의 innerHTML 속성 이용
  - ✓ body 태그에 내용을 추가하고 싶을 때
  - ✓ div 태그에 내용을 추가하고 싶을 때

```
  <script>

    function addText() {
        var div = document.getElementById("textSpace");

        div.innerHTML = "이것은 소리없는 아우성!!";
    }

  </script>
</head>
<body>
    <div id="textSpace"> </div>
    <form action="#" method="post">
        <input type="button" value="새로운 Text추가" onclick="addText()" />
    </form>
</body>
```

- 문서 객체의 style 속성 사용
  - ✓ 해당 문서 객체의 스타일 변경 가능

```
  <script>

    function addText() {
        var div = document.getElementById("textSpace");

        div.innerHTML = "이것은 소리없는 아우성!!";
        div.style.border = "2px Solid Orange";
        div.style.background = "Yellow";
        div.style.color = "Black";
        div.style.fontFamily="궁서";
    }

  </script>

</head>
<body>
    <div id="textSpace"> </div>
    <form action="#" method="post">
        <input type="button" value="새로운 Text추가" onclick="addText()" />
    </form>
</body>
```

- 문서 객체 가져오기
  - ✓ document 객체의 getElementById() 메서드
  - ✓ 한 번에 한 가지 문서 객체만 가져올 수 있음

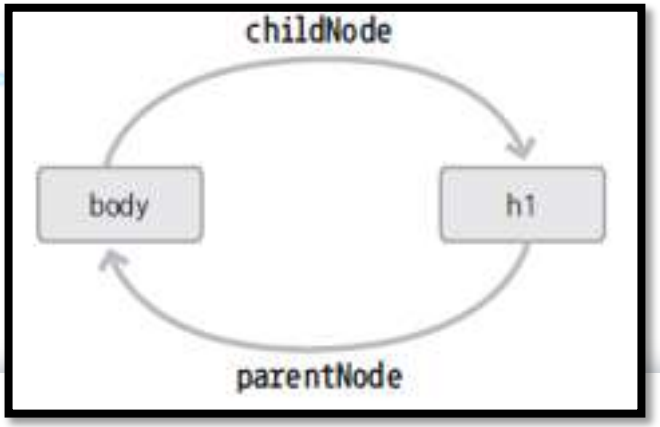
- 한 번에 여러 개의 문서 객체를 가져올 수도 있다.

메서드 이름	설명
getElementsByName(name)	태그의 name속성이 name과 일치하는 문서 객체를 배열로 가져옵니다.
getElementsByTagName(tagName)	tagName과 일치하는 문서 객체를 배열로 가져옵니다.

- 문서 객체 제거
  - ✓ 일반적인 문서 객체 제거 코드
  - ✓ willRemove.parentNode.removeChild(willRemove);

  - ✓ 삭제하려는 태그에서 부모 노드로 이동한 후
  - ✓ 부모 노드에서 자식 노드 삭제

메서드 이름	설명
removeChild(child)	문서 객체의 자식 노드를 제거합니다.



- 문서 객체 제거

```
  <script>

    function deleteDiv() {
        var div = document.getElementById("textSpace");
        div.remove();
    }

  </script>

</head>
<body>
    <div id="textSpace"> 이것은 소리없는 아우성!! </div>
    <form action="#" method="post">
        <input type="button" value="DIV 제거" onclick="deleteDiv()" />
    </form>
</body>
```

4.EVENT

JavaScript

- 이벤트란?
  - ✓ 사용자가 하는 모든 행동
  - ✓ Ex) 키보드로 키를 입력하거나 마우스 클릭
- 자바스크립트가 지원하는 이벤트
  - ✓ 애플리케이션 사용자가 발생
  - ✓ 애플리케이션이 스스로 발생
  - ✓ 마우스 이벤트
  - ✓ 키보드 이벤트
  - ✓ HTML 프레임 이벤트
  - ✓ HTML 입력 양식 이벤트
  - ✓ 유저 인터페이스 이벤트
  - ✓ 구조 변화 이벤트
  - ✓ 터치 이벤트

- 이벤트 관련 용어
  - ✓ “이벤트를 연결한다”
  - ✓ Ex) window 객체의 onload 속성에 함수 자료형 할당

- Load - 이벤트 이름 (Event Name) 또는 이벤트 타입 (Event Type)

- onload 이벤트 속성

- 이벤트 핸들러 - 이벤트 속성에 할당한 함수

- 이벤트 모델의 종류

- DOM Level 0
  - ✓ 고전 이벤트 모델
  - ✓ 인라인 이벤트 모델

- DOM Level 2
  - ✓ 표준 이벤트 모델

- 고전 이벤트 모델
  - ✓ 자바스크립트에서 문서 객체의 이벤트 속성을 사용해 이벤트 핸들러 연결

```
window.onload = function() {

    var header = document.getElementById("myH1");

    header.onclick = function() {
        alert("H1이 클릭되었어요!!");
    }
}
```

- 고전 이벤트 모델
  - ✓ 이벤트 핸들러 제거시 문서 객체의 이벤트 속성에 null 할당
  - ✓ 이벤트 하나에 이벤트 핸들러 하나

```
window.onload = function() {

    var header = document.getElementById("myH1");

    header.onclick = function() {
        alert("H1이 클릭되었어요!!");

        header.onclick = null;
    }
}
```

- 고전 이벤트 모델
  - ✓ 이벤트의 source를 밝힐 수 있음 - this 키워드 이용

```
window.onload = function() {

    var header = document.getElementById("myH1");

    header.onclick = function() {
        alert(this);
    }
}
```

- 이벤트 객체 사용
  - ✓ 이벤트 객체 내용 출력하는 예제
  - ✓ 현재 발생한 이벤트의 세부 정
    있다.
- 이벤트 핸들러의 매개 변수로 전달

```
<body>
    <input type="button" id="btn"
           value="이벤트 발생!!" />
</body>
```

```
window.onload = function() {
    var btn = document.getElementById("btn");

    btn.onclick = function(e) {
        alert("버튼 클릭 event 발생!!" + e);
        for( var i in e) {
            console.log("key : " + i +
                        ", value : " + e[i]);
        }
    }
}
```

- 이벤트 강제 발생시키는 방법
  - ✓ 메소드 호출하는 것처럼 이벤트 속성을 호출해 이벤트 강제 실행

header.onclick()

- 이벤트 강제 발생 구현 실습
  - ✓ 버튼 A 클릭하면 A의 클릭 횟수 1 증가
  - ✓ 버튼 B 클릭하면 B의 클릭 횟수는 물론 A의 클릭 횟수까지 증

```
<body>
    <input type="button" id="btnA" value="버튼 A" />
    <input type="button" id="btnB" value="버튼 B" />
    <h1>Button A Count : <span id="aCount"> </span></h1>
    <h1>Button B Count : <span id="bCount"> </span></h1>
</body>
```

- 인라인 이벤트 모델
  - ✓HTML 페이지의 가장 기본적인 이벤트 연결 방법

```
<head>
  <meta charset="utf-8" />
  <title>index</title>
  <script>
    function h1Click() {
        alert("inline event model입니다.!!");
    }
  </script>
</head>

<body>
    <h1 onclick="h1Click()"> 클릭해 보세요!!</h1>
</body>
```

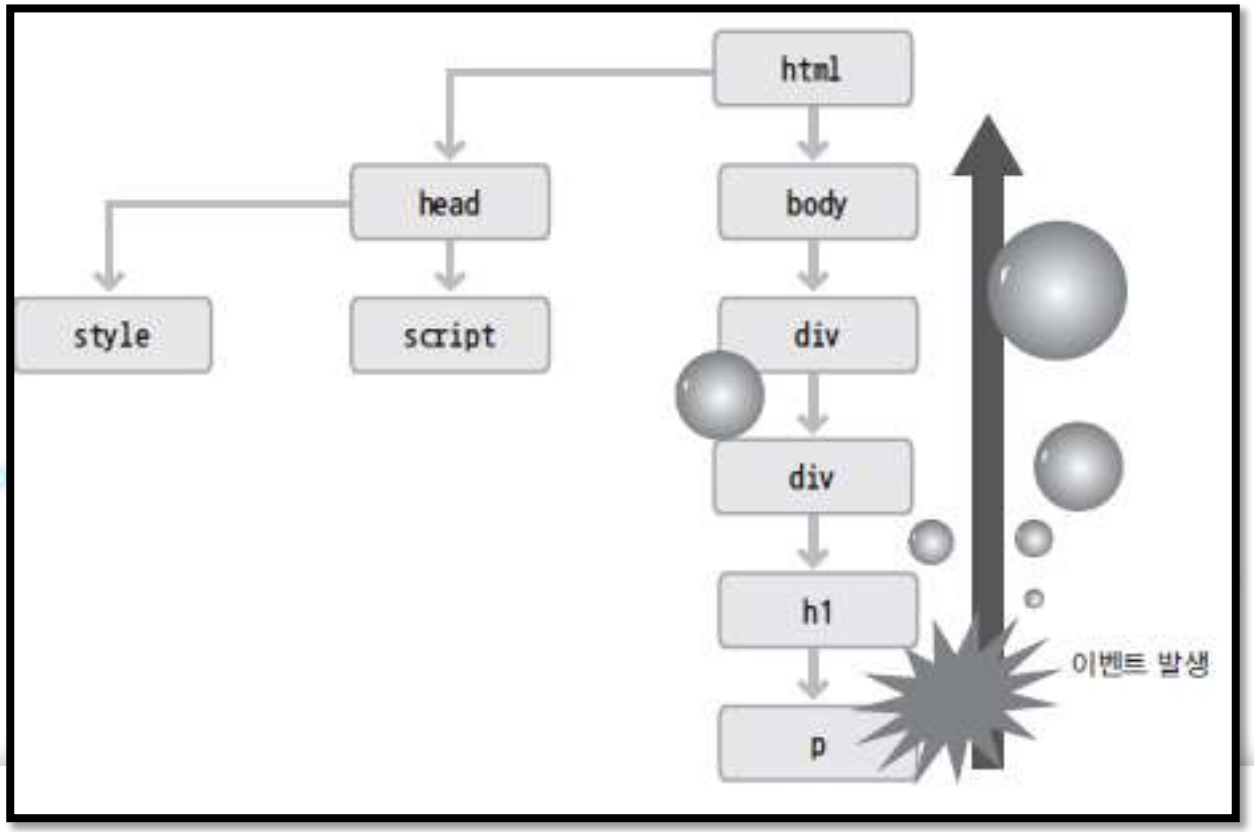
- 기본 이벤트란?
  - ✓ 일부 HTML 태그는 이미 이벤트 핸들러 가지고 있음
  - ✓ 입력양식(form)의 내용을 전송하는 경우
  - ✓ 기본 이벤트를 제거하려면 해당 이벤트 속성에 false값 전달.

```
<head>
  <meta charset="utf-8" />
  <title>index</title>
  <script src="js/defaultEventNo.js"> </script>
</head>
<body>
    <a href="http://www.google.com"
       id="goGoogle"> Google 가기</a>
</body>
</html>
```

JavaScript

```
window.onload = function() {
    var link = document.getElementById("goGoogle");
    link.onclick = function() {
        var result = confirm("정말 이동하시겠습니까?");
        if(!result)
            return false;
    }
}
```

- 이벤트 전달
  - ✓ 어떤 순서로 발생하는가?
  - ✓ 이벤트 버블링 방식이 일반적
  - ✓ 자식 노드에서 부모 노드 순으로 이벤트 실행



JavaScript

```
<head>
  <meta charset="utf-8" />
  <title>index</title>
  <script src="js/eventBubblingNo.js"> </script>
</head>
<body>
    <div id="outerBox" onclick="outerBox()">
        <img src="img/shin.png" onclick="innerImg()" />
    </div>
</body>
```

```
function outerBox() {
    alert("div click!!");
}

function innerImg() {
    alert("image click!!");
}
```

- 이벤트 전달 막기
  - ✓ 이벤트 객체의 stopPropagation() 메소드 사용

```
function outerBox() {
    alert("div click!!");
}

function innerImg() {
    alert("image click!!");
    event.stopPropagation();
}
```

- 기본이벤트 전달 막기
  - ✓ 이벤트 객체의 preventDefault() 메소드 사용

```
<body>
    <div id="outerBox" onclick="outerBox()">
        <a href="http://www.google.com">
            <img src="img/shin.png" onclick="innerImg()" />
        </a>
    </div>
</body>
```

```
function outerBox() {
    alert("div click!!");
}

function innerImg() {
    alert("image click!!");
    event.preventDefault();
}
```

- DOM Level 2
  - ✓ 인라인 이벤트 모델, 고전 이벤트 모델의 단점 - 한 번에 하나의 이벤트 핸들러만 가질 수 있음

- 표준 이벤트 모델
  - ✓ 웹 표준 단체인 W3C에서 공식 지정한 DOM Level 2 이벤트 모델
  - ✓ 한 번에 여러 가지 이벤트 핸들러 추가 가능

  - ✓ addEventListener(eventName, handler)
  - ✓ removeEventListener(eventName, handler)

JavaScript

```
<head>
  <meta charset="utf-8" />
  <title>index</title>
  <script src="js/standardEventModel.js"> </script>
</head>
<body>
    <div id="outerBox">
        <a href="http://www.google.com" id="anchor">
            <img src="img/shin.png" id="img" />
        </a>
    </div>
</body>
```

JavaScript

```
window.addEventListener("load",winLoad);
function winLoad() {
    var div = document.getElementById("outerBox");
    var anchor = document.getElementById("anchor");
    var img = document.getElementById("img");

    div.addEventListener("click",outerBox);
    img.addEventListener("click",innerImg);
    img.addEventListener("click",stopEvent);
}
function outerBox() {
    alert("div click!!");
}
function innerImg() {
    alert("image click!!");
}
function stopEvent() {
    event.stopPropagation();
    event.preventDefault();
}
```

THANK YOU

Brendan Eich, CTO, Mozilla Corporation