화면 구현

목 차

Part 1. UI 설계 확인하기

Chapter 1. UI 설계 내용 확인하기

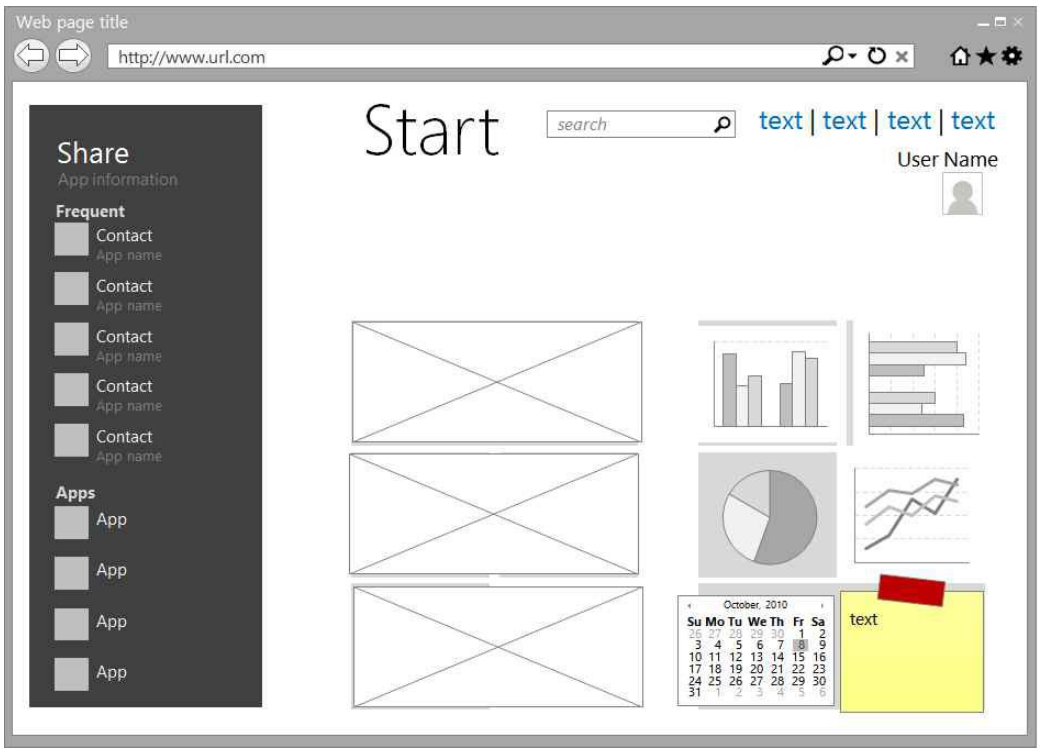
1. UI설계서

웹 사이트의 페이지 구성요소를 기록한 설계도이다. 정적인 형태의 화면 형태로 와이어프레임이나 목업 등을 이용하여 작성한다.

2. UI설계서의 이해를 위해 UI설계 도구의 종류와 특징을 확인한다.

2-1 UI 설계 도구들의 종류를 조사한다.

(1) UI설계서의 종류에 따라 내용 구성이 다르므로 설계 도구에 대한 종류 파악이 필요하다.



[그림 1-1] 화면 설계서 예시

(2) 화면 설계서의 화면 구성(레이아웃) 측면에서 내용을 점검한다.

(가) 메뉴 구성과 사용된 이미지 등이 웹 페이지의 특성에 적절한지 체크한다.

(나) 화면을 구성하는 문자열이나 내용이 적절한지 체크한다.

(다) 화면 구성요소들이 화면에서 조화를 이루는지 체크한다.

(라) 화면을 구성하는 항목들이 표현하는 색상과 균형감이 적절한지 체크한다.

2-2 UI설계서에 포함되어야 할 항목들을 확인한 후 요구사항도 확인한다.

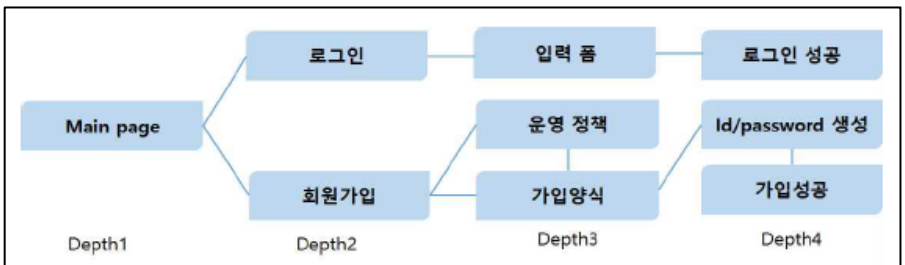
1. UI설계서에 화면 구현 단계에서 반영되어야 할 요구사항과 지켜야 할 제한사항 등을 확인하여 포함시킨다.

<표 1-1> 내 요구사항 명세서 예시

번호	기능	요구사항	요청자
1	인터페이스	최대한 간편하게 구성한다.	
2	인터페이스	검색 창을 눈에 띄게 배치한다.	
3	검색	수동 검색이 가능하도록 한다.	
4	검색	전체와 부분 검색이 가능하도록 한다.	
5	출력	사용자의 요구에 맞게 검색 결과를 출력하도록한다.	
6	메뉴	하부 메뉴로 이동이 편리하도록 설계한다.	

2-3 UI설계 내용을 분석하여 전체적인 화면과 폼의 흐름을 확인한다.

1. 메인 페이지에서 로그인과 회원가입 메뉴의 구조를 살피고 메뉴의 흐름이 원활하게 이루어지는지 확인한다.

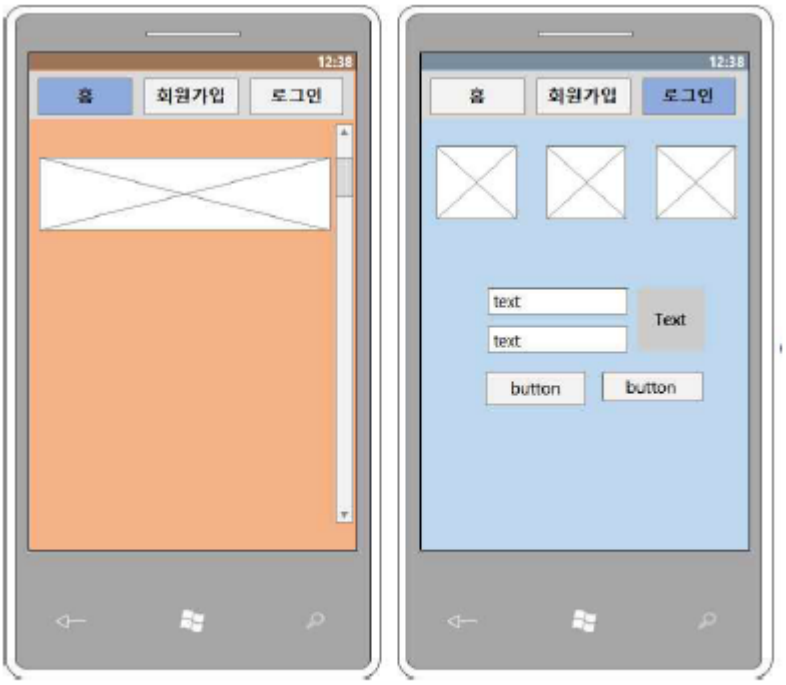


[그림 1-2] 화면 구성의 계층적 구조 예시

2. 메인 화면에서 메뉴의 위치 및 배열을 확인하고, 버튼을 선택하였을 때 나타나는 화면에서 필요한 화면이 나타나는지 등의 계층적인 화면 연결의 흐름을 확인한다.

   다음 [그림 1-2],[그림 1-3] 로그인/회원가입 화면 흐름도 예시 자료와 같이 메인 화면에서의 로그인과 회원가입 메뉴의 위치 및 배열을 확인하고, 로그인 버튼을 선택하였을 때 나타나는 화면에서 로그인 정보를 입력 받을 수 있는 화면이 나타나는지 등의 계층적인 화면 연결의 흐름을 확인한다.

   로그인 / 회원가입 기능을 수행하는 화면 전환의 기능에 사용자가 요구하는 사항이 포함되어 있는지, 화면 구성에 필요한 모든 요소가 포함되어 있는지 확인한다.



[그림 1-3] 홈 화면과 로그인 화면 예시

3. UI설계 내용을 확인하면서 다음과 같은 내용을 확인한다.

<표 1-2> UI설계 확인 내용 예시

번호	확인 내용	여부
1	UI가 사용자 측면에서 설계되고 사용자 편리성이 반영되었는가?	
2	화면 이동이 쉽게 설계되었는가?	
3	사용자가 요구하는 내용을 직관적으로 파악할 수 있도록 되어 있는가?	
4	논리적인 메뉴 구조를 갖추고 있으며 메뉴의 의미 전달이 정확하게 되어 있는가?	
5	화면에 표시할 정보, 화면구성, 화면 간 흐름 등에 대한 표시가 잘 나타나 있는가?	
6	UI는 로그인/로그아웃 버튼의 일반적인 경우처럼 화면을 구성하는 항목들이 보편적인 위치에 지리하고 있는가?	

4. <표 1-2>에 제시된 것과 같은 설계 확인 내용을 확인하고 화면 흐름의 확인 과정에서 오류가 발견될 경우 설계 내용이 수정될 수 있도록 설계 팀과 협의한다.

Chapter 2. UI 메뉴 구조 확인하기

1. UI 메뉴 구조 확인

1-1 화면 구현 단계에서 고려되어야 할 모든 요구사항을 정리한다.

(가) 시스템 개발 절차와 관련된 개발 일정, 비용 등을 확인한다.

(나) 요구사항에 포함된 기능들이 표현되어 있는지 확인한다.

(다) 요구사항과 설계 내용을 비교하여 추가되어야 할 내용들이 없는지 확인한다.

(라) 화면에 필요한 요소들이 사용자 편리성을 고려하여 배치되어 있는지 확인한다.

(마) 요구사항 명세서와 제약사항에 포함된 항목(기능,성능,인터페이스,표준지침 등) 등이 포함되었는지 확인한다.

번호	확인 내용	여부
1	요구사항이 동일한 의미로 일관성 있게 제시되고 있는가?	
2	사용자가 요구하는 기능 등이 모두 포함되어 있는가?	
3	사용자의 요구사항들이 기능적인 면이나 내용적인 면에서 서로 충돌되는가?	
4	내용이 객관적이고 구체적인가?	
5	일정,비용,구현 등이 가능한가?	

1-2 UI표준과 제약사항에 관련된 내용은 정리하여 숙지하도록 한다.

(가) 개발 팀과 디자인 팀 등 개발 관련 팀 모두가 화면 구현 과정에서 항상 참고할 수 있도록 정리된 자료를 공유한다.

(나) 공유된 자료는 소프트웨어 개발 과정에서 표준과 지침의 범주에서 벗어나지 않도록 UI표준과 지침을 숙지하도록 한다.

1-3 설계 단계에서 UI표준과 지침이 반영되어 화면 설계가 이루어졌는지 확인하고, 화면 구현을 위한 기본 사항을 점검한다.

(가) 사용자의 편리성과 업무의 효율성이 고려되었는지 확인한다.

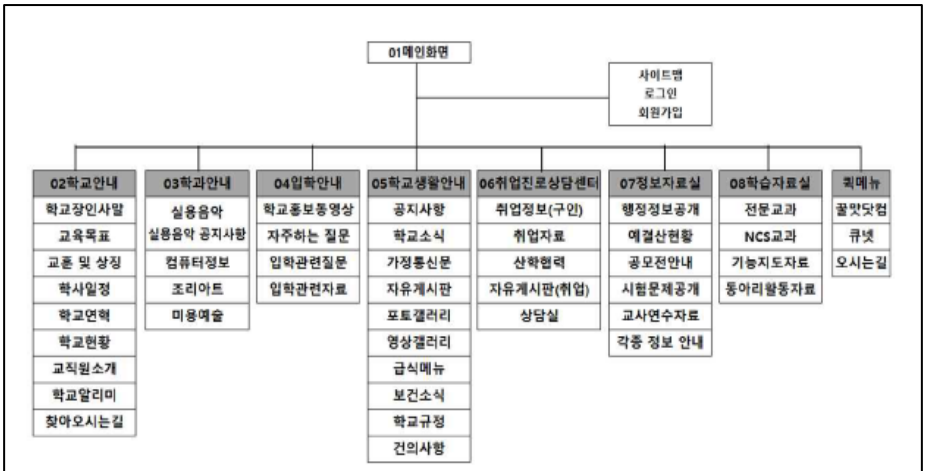
(나) 화면을 구성하는 각 페이지마다의 요소(로그인/로그아웃 등)들이 통일성이 유지되도록 설계되었는지 확인한다.

1-4 전체 사이트의 구조와 흐름을 파악하여 사이트 맵을 작성한다.

(가) 사이트 메뉴 구조는 내비게이션을 설계하기 위해 사용되며, 페이지의 연결 관계나 계층적 구조와 흐름을 파악한다.

(나) 사이트에 포함된 정보와 화면의 흐름을 알 수 있도록 메뉴별 하위 페이지와 메뉴 구조 작성은 정보 구조 표현 기법을 이용한다.

(다) 사용자 요구사항 명세서와 화면 설계를 기준으로 사이트 메뉴 구조도를 작성한다.



[그림 1-7] 사이트 메뉴 구조도

1-5 작성된 사이트 메뉴 구조를 검토한다

(가) 메뉴 구조를 보고 사용자의 요구사항 중 누락된 내용이 없는지 확인한다.

(나) 메뉴의 연계성과 화면의 흐름이 UI표준과 지침을 준수하고 있는지 확인한다.

(다) 하위 페이지 메뉴에 대한 메뉴 구조도를 작성하여 화면 구현 프로그램 작성 단계에서 자료로 활용한다.

1-6 화면 설계서와 메뉴 구조를 참고하여 화면 설계 내용을 확인한다.

(가) 메인 페이지나 하위 페이지 등에 포함된 메뉴나 폼이 사용자와 시스템과의 원활한 소통이 이루어지도록 설계되었는지 확인한다.

(나) 사용자가 원하는 정보의 검색이 가능하도록 설계가 이루어졌는지 세부 항목 및 부메뉴를 점검한다.

(다) 화면과 폼에 명시된 화면 이미지와 상세 설명을 정확하게 숙지하여 하위 페이지와의 링크 관계를 파악한다.

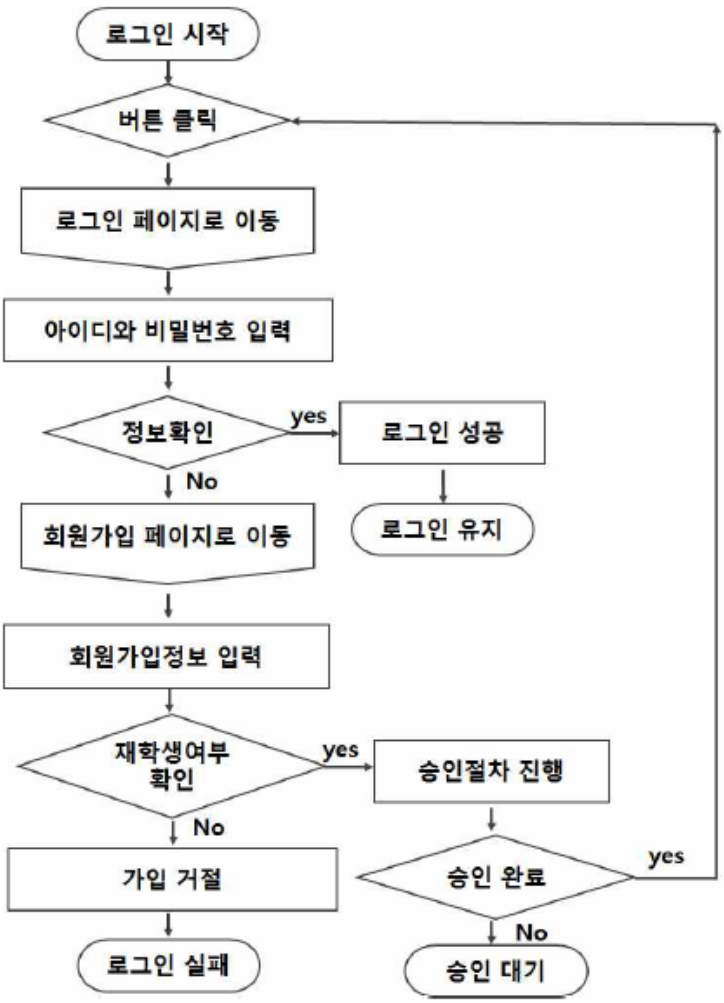
(라) 화면 설계 내용에 오류가 있는 경우 설계 담당자와 협의한다.

1-7 화면 설계 내용을 확인하고 폼의 흐름을 파악하여 구현 절차에 따라 작성한다.

(가) 화면을 구현하기 위한 절차 화면을 구성하는 기능들에 대한 요구사항과 세부 처리 절차를 포함하고 있는 유스케이스를 작성한다.

유스케이스명	로그인	
개요	학교 홈페이지에 접속하기 위한 로그인 과정을 수행한다.	
관련 액터	학생	
선행 조건	재학생 정보 목록이 있는 상태	
흐름 (시나리오)	기본	1. 학생은 메인 화면에서 "로그인"을 선택한다. 2. 시스템은 로그인 화면을 보여준다. 3. 학생은 로그인 메인화면에서 자신의 아이디와 비밀번호를 입력하고 "로그인"을 선택한다. 4. 시스템은 아이디와 비밀번호가 회원 정보 목록에 있는지 확인한다. 5. 학생의 아이디가 정보 목록에 있는 경우 로그인 상태로 전환한다. 6. 로그인 메뉴는 "로그아웃"으로 변경한다.
	대안	4-1. 학생의 아이디가 회원 목록에 없을 경우 1. 시스템은 로그인 불가 메시지를 발생한다. 2. 회원가입 화면을 표시한다. 3. 학생이 가입에 필요한 정보를 입력하고 가입을 선택한다. 4. 시스템은 학생 정보 목록에서 학생의 인적 사항을 확인한다. 5. 학생의 인적 사항이 있는 경우 가입 인증 절차를 거쳐 회원 정보를 목록에 자료를 추가한다. 6. 학생의 인적 사항이 학교의 재학생이 아닌 경우 가입 거절 메시지를 나타낸다.
기타 요구사항	성능 : 일반적인 네트워크 로드 하에서 10회 중 7회 이상이 3초 내에 로그인 상태로 전환되어야 한다.	

유스케이스를 참고하여 화면을 구현하기 위한 절차도를 작성한다.



[그림 1-8] 로그인 화면 구현 절차도

(나) 앞 단계에서 작업한 내용을 토대로 다른 화면의 구현에 필요한 절차도를 작성하면서 화면 설계 내용에 대한 확인 작업을 마치도록한다.

(다) 단계별 프로그램을 구현할 때 주의해야 할 상항을 정리하여 개발 팀과 디자인팀 간에 자료를 공유한다.

Part 2. UI 구현하기

Chapter 3. JavaScript

1. 자바스크립트 개요

자바스크립트(JavaScript)는 객체 기반의 스크립트 프로그래밍 언어이다. 이 언어는 웹브라우저 내에서 주로 사용하며, 다른 응용 프로그램의 내장 객체에도 접근할 수 있는 기능을 가지고 있다. 또한 구글에서 개발한 Node.js와 같은 런타임 환경과 같이 서버 사이드 네트워크 프로그래밍에도 사용되고 있다.

자바스크립트는 본래 넷스케이프 커뮤니케이션즈 코퍼레이션의 브렌던 아이크(Brendan Eich)가 처음에는 *모카*(Mocha)라는 이름으로, 나중에는 *라이브스크립트*(LiveScript)라는 이름으로 개발하였으며, 최종적으로 자바스크립트가 되었다.

자바스크립트가 썬 마이크로시스템즈의 자바와 구문(syntax)이 유사한 점도 있지만, 이는 사실 두 언어 모두 C 언어의 기본 구문을 바탕했기 때문이고, 자바와 자바스크립트는 직접적인 관련성이 없다. 이름과 구문 외에는 자바보다 셀프와 유사성이 많다.

2013년 1월 기준으로, 가장 최근 버전은 자바스크립트 1.8.5이고, 파이어폭스 3에서 지원된다. 표준 ECMA-262 3판에 대응하는 자바스크립트 버전은 1.5이다. ECMA스크립트는 쉽게 말해 자바스크립트의 표준화된 버전이다. 모질라 1.8 베타 1이 나오면서 XML에 대응하는 확장 언어인 E4X(ECMA-357)를 부분 지원하게 되었다. 자바스크립트는 브라우저마다 지원되는 버전이 다르며, 가장 범용적으로 지원되는 버전은 1.5이다.

2. 자바스크립트 사용방법

2.1 head / body 태그 영역 안에 script 태그 삽입하기

HTML 문서에 삽입하여 사용하는 방식이다.

<head> 태그나 <body>태그 영역안에 <script></script> 태그를 삽입하고 자바스크립트 구문을 입력해서 실행되게 한다. <head> 태그에는 주로 문서가 로딩될 때 미리 실행할 내용을 작성하고, <body> 태그는 특정 부분에서 실행될 내용을 작성한다.

```
<html>
<head>
<script language="javascript">
<!--
window.alert("Head 영역에 작성한 자바스크립트 코드입니다.");
```

```
//-->
</script>
</head>
<body>
<script language="javascript">
<!--
document.write("Body 영역에 작성한 자바스크립트 코드입니다.");
//-->
</script>
</body>
</html>
```

위 소스의 <!-- //--> 는 HTML 문서의 주석코드를 자바스크립트 코드에 삽입한 것이며 자바스크립트 코드의 주석으로 처리되지 않는다. 구 버전의 브라우저나 몇몇 특정 브라우저에는 자바스크립트 엔진이 없을 경우 자바스크립트 코드를 인식하지 못하고 에러를 발생시키게 되는데 이를 방지하기 위하여 삽입한 것이다. 하지만 최근의 주류 브라우저들은 모두 자바스크립트 엔진을 내장하고 있기 때문에 굳이 사용하지 않아도 된다.

자바스크립트 코드는 별도의 컴파일이 필요없으며 HTML 문서 내에 삽입되어 소스가 공개된다..

2.2 HTML 태그 on이벤트 속성에 간단한 코드 직접 작성하기.

```
<html>
<head>
</head>
<body>
  <button onclick="javascript: window.alert('hello, javascript');">버튼</button>
</body>
</html>
```

button 태그를 생성하고, 태그안 onclick 속성에 간단한 메시지창을 출력하는 자바스크립트 코드를 작성하였다. 해당 파일을 브라우저에서 실행하고, 버튼을 클릭하면, 메시지 창에 hello, javascript 문구를 확인할 수 있다..

2.3 외부 자바스크립트 파일(*.js) 가져오기

우선 sample.js 스크립트 파일 해당 html 문서 파일과 같은 폴더에 아래의 소스 코드를 작성하고 저장한다.

window.onload 이벤트에 이벤트 핸들러를 연결 작성한다.

```
window.onload = function(){
    alert("sample.js 에서 실행하는 스크립트입니다.");
}
```

이제 아래와 같이 html 파일에서 script 태그의 src 속성에 ./sample.js 라고 기술한다.

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

3. 자바스크립트 데이터 타입

자바스크립트에서는 Number, String, Boolean, Function, Object, Null, undefined, Array 등의 데이터 타입이 존재한다. Function은 function 타입의 일종이고, Array, Date, RegExp 같은 타입은 Object 타입의 일종이다.

우선 typeof 연산자를 통해 선언된 변수의 데이터 타입을 체크해 보자.

```
 var str1 = "abcdef";
var str2 = 'abecef';
 var num = 1234;
var bool = true;
var obj = {"abcdef":123};
var arr = [1, 2, 3, 4];

var val;
var el = document.getElementById("notExist");  //존재하지 않는 엘레먼트

var regExp = /^[0-9]*$/;     // 숫자를 검사하는 정규표현식
var date = new Date();       //오늘날짜를 담은 날짜객체 생성.
var f = function( ) {
    alert("hello, js~")
}

console.log("str1 => "+ typeof str1);
console.log("str2 => "+ typeof str2);
console.log("num => "+ typeof num);
console.log("bool => "+ typeof bool);
console.log("obj => "+ typeof obj);
console.log("arr => "+ typeof arr);
console.log("val => "+ typeof val);
console.log("el => "+ typeof el);
console.log("regExp => "+ typeof regExp);
console.log("date => "+typeof date);
console.log("f => "+ typeof f);
```

결과

```
// str1 => string
// str2 => string
// num => number
// bool => boolean
// obj => object
// arr => object
// val => undefined
// el => object
// regExp => object
// date => object
// f => function
```

3.1 문자열(String)

문자열 값을 표현하는데 사용하는 데이터 타입으로, 16비트 유니코드 문자의 연결 구조이다
객체로 취급된다. (property(속성)과 method(기능)를 가진다)

프로퍼티	설명
length	Stirng 개체 길이 반환

메소드	설명
charAt()	지정한 인덱스에 해당하는 문자를 반환
replace()	정규식을 사용하여 문자열의 텍스트를 바꾸고 결과 반환
toUpperCase()	모든 영문자가 대문자로 변환된 문자열을 반환.
toLowerCase()	모든 영문자가 소문자로 변환된 문자열을 반환
substring()	string 객체 안의 지정된 위치에 있는 부분 문자열을 반환
split()	문자열을 구분자를 사용해 분할하고, 문자열 배열에 담아 반환.

length property

문자열의 길이를 반환하는 String 객체의 프로퍼티이다.

```
<script>
    console.log("hello".length);
</script>
```

실행결과> 5

console.log( ) 의 결과는 브라우저의 개발자 도구(단축키는 주로 F12) 내의 콘솔(console) 창을 통해 확인할 수 있다.

charAt(), replace(), toUpperCase(), split() method

String타입에서 자주 사용되는 메소드이다.

```
<script>
console.log("hello world".charAt(0));       // 특정 인덱스 위치의 문자를 반환
</script>
실행결과> h
```

```
<script>
console.log("hello world".replace("hello", "hi"));       // 기존 문자열을 다른 문자열로 치환
```

```
</script>
실행결과> hi world
```

```
<script>
console.log("hello world".toUpperCase());          // 문자열을 대문자로 변환
</script>
실행결과> HELLO WORLD
```

```
<script>
console.log("1,2,3,4,5".split(","));  // ","를 토큰으로 문자열을 나누어 문쟈열 배열을 반환
</script>
실행결과> ["1", "2", "3", "4", "5"]
```

1.2 숫자(Number)

숫자를 표현하거나 산술 연산에 사용되는 데이터 타입이다.

Math라는 내장객체를 이용하여 수학함수의 결과를 숫자타입으로 얻을 수도 있다

Number 캐스팅(Casting) – parseInt(), parseFloat()

내장함수 중 parseInt(), parseFloat()을 사용하면 숫자문자열을 쉽게 숫자로 변환 가능하다

```
<script>
console.log(parseInt("010", 10));          // 두번째 인자로 진법을 지정한다
</script>
실행결과> 10        // 10진수로 "010"을 변환
```

```
<script>
console.log(parseInt("010", 2));           // 이진수로 지정
</script>
실행결과> 2
```

단, 두 번째 인자를 지정하지 않을 경우 숫자 앞에 0이 붙어있으면 8진수, 0x가 붙어있으면 16진수로 인식한다

```
<script>
    console.log(parseInt("010"));                  // 숫자 앞에 0이 있으면 8진수로 인식
</script>
실행결과> 8
```

parseInt() 함수는 소수점 이하 자리를 버리므로 소수점을 유지하고 싶다면 parseFloat() 함수를 사용해야 한다

```
<script>
    console.log(parseFloat("010"));                    // 진법을 지정하지 않아도 10진수로 반환
</script>
실행결과> 10
```

```
<script>
  console.log(parseFloat("03.1415"));  // 진법을 지정하지 않아도 10진수로 반환 및 소수점 유지
</script>
실행결과> 3.1415
```

NaN – Not a Number

문자열을 숫자로 데이터 형변환시에 문자열이 숫자가 아닐경우 Not a Number의 약자인 NaN을 리턴하며 숫자의 형태가 아니라는 의미이다.

- NaN의 타입은 number이다.
- Infinity 끼리의 연산은 NaN을 리턴한다
- 음수의 제곱근인 허수는 자바스크립트에서 표현불가하다. 그러므로 NaN이다.

```
<script>
    var a = 0/0;
    var b = "food"*1000;
    var c = Math.sqrt(-9);//음수의 제곱근은 허수이다. 자바스크립트에선 허수표현불가!!

    Console.log("a => "+a);
    console.log("typeof a => "+typeof a);
     console.log("b => "+b);
    console.log("c => "+c);

    var d = 10/0;// Infinity
    var e = d – d;// Infinity-Infinity
    console.log("e => "+e); // e => NaN
    console.log("typeof e => "+typeof e);    // typeof a => number
</script>

> a => NaN
```

```
> typeof a => number
> b => NaN
> c => NaN
> e => NaN
```

IsNaN()

NaN(Not a Number) 인지 검사하는 내장함수이다.

Number 형이 아니라면 true, number 형이 맞다면 false를 리턴한다

```
<script>
var x = parseInt("hello",10);
    if(isNaN(x)){
        document.write("변수 x는 NaN입니다.<br>");
    }

    var y = 9876;
    if(isNaN(y)){
        document.write("변수 x는 NaN입니다.");
    }
    else {
        document.write("변수 x는 NaN이 아닙니다.");
    }
</script>
```

변수 x는 NaN입니다.
변수 x는 NaN이 아닙니다.

빈 문자열, 공백만 있는 문자열을 isNaN() 함수를 통해 검사하면, false가 나오므로, 유효성 검사시 주의하자.

```
<script>
  //빈문자열은 NaN이 아니다.
    Var k = ""; //빈문자열
    var l = ' '; //공백만 있는 문자열
```

```
    console.log(isNaN(k));
    console.log(isNaN(l));
</script>
>false
>false
```

Infinity, -Infinity, isFinite()

양의 무한대의 숫자를 나타내는 Infinity, 음의 무한대의 숫자를 나타내는 –Infinity

```
<script>
   console.log(1/0);
</script>
실형결과> Infinity
```

```
<script>
   console.log(-1/0);
</script>
실행결과> -Infinity
```

무한대가 아닌 숫자인지 판별하는 내장함수 isFinite()

```
<script>
     console.log(isFinite(1/0);
</script>
실행결과> false
```

```
<script>
     console.log(isFinite(1));
</script>
실행결과> true
```

1.3 널(null)과 undefined

널(null)은 객체가 존재하지 않음을 나타낸다

undefined는 초기화되지 않았거나 선언되지 않았거나 값이 할당되지 않았음을 나타낸다

- null의 타입은 객체이다.
- undefined의 타입은 undefined이다.

```
Var test = null;
console.log("typeof test => "+ typeof test); // typeof test => object

//정의되지 않았거나, 값이 할당되지 않은경우
var val1;
console.log("val1 => "+ typeof val1);
console.log("val2(정의된적없음) => "+ typeof val2);
```

```
typeof test => object
val1 => undefined
val2(정의된적없음) => undefined
```

3.4 Boolean

논리값을 표현하는 데이터 타입이다. true와 false값을 가진다
true / false 값을 통해 표현식의 참과 거짓을 구분할 수 있다.

```
<script>
    console.log(123 < 1000);
</script>
```

실행결과> true

자바스크립트에서 데이터타입은 조건식에서 true / false로 치환된다. false로 처리되는 데이터 값은 다음과 같다.

- undefined
- null
- 0
- ""
- NaN

```
<script>
    //undefinde, null, 0, "", NaN
    var test_undefined;
    if(test_undefined){
        //test_undefined 는 undefined의 값을 가지고, 조건식에서 false로 처리된다.
    }
```

```
        var element = document.getElementById("notExist");
        if(element){
            //element는 null값을 가지고, 조건식에서 false로 처리된다.
        }

        if(0){
            //0은 조건식에서 false로 처리된다.
        }

        if(""){
            //"" 빈 문자열은 조건식에서 false로 처리된다.
        }

        if(NaN){
            //NaN은 조건식에서 false로 처리된다.
    }
```

그 외 데이터들은 모두 true로 처리된다.

```
    <body>
    <span id="exist"></span>
    <script>
        //유사 참
        if([]){
            //빈 객체 object는 참이다.
        }

        element = document.getElementById("exist");
        if(element){
            element.innerHTML = "존재하는 엘레먼트는 참이므로 이것이 실행된 것이다..."
        }

        if(1){
            //0이 아닌 모든 숫자는 참이다.
```

```
        }

        var str = "빈 객체열 아님";
        if(str){
            //str은 참이다.
    }
</script>
</body>
```

존재하는 엘레먼트는 참이므로 이것이 실행된 것이다...

4 배열

배열이란 자료형이 일치하는 여러 개의 데이터를 냐열하여 저장하고 사용할 수 있다
서로 연관 있는 데이터를 하나의 배열에 담아 관리한다

4.1 배열 생성

생성자를 사용하거나, 리터럴로 배열을 선언할 수 있다. Array 생성자를 사용할 때는 자바와 달리, 배열의 크기는 소괄호안에 기입해야 한다. 배열의 크기는 동적으로 할당되므로, 배열의 인덱스 크기 이상도 지정 가능하다.

```
<script>
    //배열선언1 : 생성자 호출
     var arr1 = new Array(3);
    arr1[0] = "a";
    arr1[1] = "b";
    arr1[2] = "c";
    console.log("arr1 = ", arr1);

    //배열선언2 : 생성자를 이용한 초기화
    var arr2 = new Array("l", "m", "n");
    console.log("arr2 = ", arr2);
```

```
    //배열선언3 : 리터럴로 바로 초기화
    var arr3 = ["x", "y", "z"];
    console.log("arr3 = ", arr3);
  </script>
```

결과

arr1 = [a,b,c]

arr2 = [l,m,n]

arr3 = [x,y,z]

배열의 인자를 할당할 때, 인덱스값을 이용해 지정하거나, 배열의 push(인자값) 메소드을 통해 할당할 수 있다

```
<script>
    var arrTest = [];
    arrTest[0] = "k";
    arrTest.push("h");
    console.log("arrTest= ["+arrTest+"]");
</script>
```

결과

arrTest= [k,h]

4.2 배열 요소 접근

선언된 배열의 데이터에 접근하기 위해서는 인덱스를 [ ] 안에 적어 접근한다. (배열명[인덱스])

0번째 인덱스부터 사용할 수 있으며 배열이 가지고 있는 데이터의 개수를 넘어 인덱스를 지정할 수 없다. (0 <= index < 배열크기)

5 Object

Array, Function, Date, RegExp, null 과 같은 데이터는 엄밀히 따지면 Object이다. 그리고 Primitive type의 wrapper인 Booean, Number, String 까지 모두 object 이다.

자바스크립트를 객제기반 언어라고 하는 이유이기도 하다. 실직적으로 자바스크립트에서 사용되는 대부분의 데이터 타입은 객체로 존재하며 그에 따른 사용 또한 객체기반이 될 수 밖에 없다.

```
<script>
    var el = document.getElementById ("notExist"); //존재하지 않는 엘리먼트이면 null 반환
    var regExp = /^[0-9]*$/;
    var date = new Date();
    var f = function(){
            alert("hello, js~");
    };

        console.log("el => "+ typeof el);
        console.log("regExp => "+ typeof regExp);
        console.log("date => "+typeof date);
       console.log("f => "+ typeof f);</script>
```

결과

```
el => object
regExp => object
date => object
f => function
```

객체는 키와 값 쌍으로써 구성된다. 객체는 리터럴로써, 혹은 객체타입의 최고조상의 Object 객체를 new 연산자로 호출함으로써 생성할 수 있다.

```
<script>
    var obj = {
      attr1 : "hello",
      attr2 : "javascript"
    };
    var obj_  = new Object();
    obj_.attr1 = "I'm a object.";

    console.log(obj);
    console.log(obj_);
    console.log(obj.attr1);
    console.log(obj.attr2);
    console.log(obj_[attr1]);
;</script>
```

```
결과
    {attr1: "hello", attr2: "javascript"}
    {attr1: "I'm a object."}
    hello
    javascript
    I'm a object.
```

6 변수

변수를 선언하여 데이터를 저장할 수 있는 공간을 만들어 사용할 수 있다

변수 만들기

변수를 만들기 위해서 'var' 키워드를 사용한다. 자바스크립트에서는 변수를 선언할 때 데이터 타입을 따로 명시하지 않는다

- 변수 선언방법

```
<script>
// 변수 선언
var 변수이름;
</script>
```

- 변수 선언하기

```
<script>
// 변수 선언
var num;
var name
var age;
</script>
```

- 변수 선언 후 값 할당하기

```
<script>
// 변수 선언
var num;
var name;
```

```
var age;

// 변수에 값 할당
num = 123;
name = "Alice";
age = "45";
</script>
```

- 변수 선언 동시에 초기화하기

```
<script>
// 변수 선언하고 초기화
var num = 111;
var name = "Bob";
var age = "20";
</script>
```

여러 개의 변수 한번에 만들기

여러 개의 변수를 한번에 선언하기 위해서는 ,(콤마)로 변수 이름을 구분하여 선언한다

```
<script>
// 변수 여러 개 선언
var 변수이름1, 변수이름2, 변수이름3, ...;
</script>
```

- 변수 여러 개 선언하고 초기화하기

```
<script>
// 변수 선언
var num = 423, name = "Cherry", age = 23;
</script>
```

변수명 지정시 주의 사항

1. 숫자로 시작하면 안 된다
2. 대소문자를 구분한다
3. 키워드를 변수명으로 사용할 수 없다
4. 일반적으로 변수명은 소문자로 시작하여 선언하는 것이 좋다
5. 상수로 사용될 변수는 모두 대문자로 선언하는 것이 좋다

변수 값 출력 확인하기

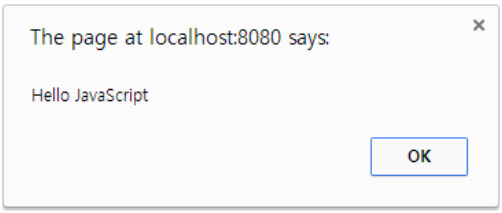
alert() 메소드 사용

특정 정보를 메시지 창을 통해 사용자에게 알려주기 위해 사용한다.

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

- 크롬브라우저에서 alert 확인

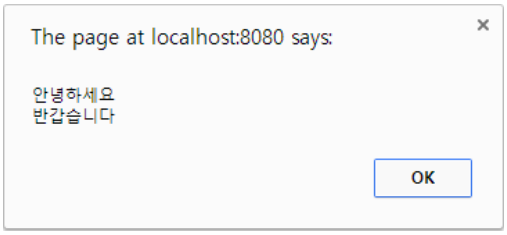
```
<script>
    alert("Hello JavaScript");
</script>
```



- 변수에 저장된 값 출력

```
<script>
    var txt = "안녕하세요.₩n반갑습니다.";
    alert(txt);
</script>
```

-



document.write() 메소드 사용

HTML 문서의 <body> 영역에 내용을 출력한다

```
<script>
    document.write(데이터);
</script>
```

console.log() 메소드 사용

브라우저의 디버깅 기능(개발자 도구)의 화면 콘솔뷰에 출력 한다

```
<script>
console.log(데이터);
</script>
```

7 주석

작성한 코드에 설명을 달아놓거나 코드가 실행되지 않도록 할 때 사용한다.
한 줄 주석과 여러 줄 주석이 있다

- 한 줄 주석

```
<script>
// 주석 처리된 구문
</script>
```

- 여러 줄 주석

```
<script>
/* 주석 처리된
   여러줄 구문 */
</script>
```

8 형변환

숫자를 문자로 변환하거나 문자를 숫자로 변환하는 것처럼 데이터 타입을 바꾸는 것을 말한다.
형변환에는 암(묵)시적 형변환과 명시적 형변환이 있다

암(묵)시적 형변환

자바스크립트 엔진이 필요에 의해 자동으로 데이터형을 변환 시키는 것이다.

비교연산시

상황	설명	예
숫자 == 문자열 비교	문자열을 숫자로 변환후 숫자와 비교.	99 == "맹구" 99 == NaN ➔ false
불린 형 == 다른형 비교	불린형을 숫자로 변환. true → 1, false → 0	1 == true 1 == 1 ➔ true
		"1" == true "1" == 1(좌항 문자열을 숫자형으로 변환) 1 == 1 ➔ true

산술연산시

덧셈

문자형과 다른 데이터형의 더하기 연산은 문자열 연결로 처리된다. 불린형과 숫자형의 더하기 연산은 true → 1, false → 0로 형변환 후 연산한다.

상황	결과	예
숫자 형 + 문자 형	문자 형	var a= 10 + "10"; // a는 문자 "1010"
불린 형 + 문자 형	문자 형	var a = true+ "10"; // a는 문자 "true 10"
불린 형 + 숫자 형	숫자 형	var a = true + 10; // a는 숫자 11

곱셈, 나눗셈, 뺄셈

숫자로 이루어진 문자형은 연산시 자동 숫자형으로 변환 후 연산한다.

상황	결과	예
숫자 형 * 문자 형	tntw 형	var m = 3 * "4"; // m = 12
숫자 형 / 문자 형	문자 형	var d = 80 / "10" // d = 8
문자 형 - 숫자 형	숫자 형	var s = "10" – 5 // s = 5

위 내용의 형변환에 대한 예제코드를 작성하고, 결과값을 예측해 본후, 화면출력을 통해 확인한다.

```
<script>
    var order1 = 1+2+"피자";
    var order2 = (1+2)+"피자";
    var order3 = 1+(2+"피자");

    var test1 = Infinity-1;
    var test2 = "54"+46;
    var test3 = 2+"1 1";
    var test4 = 99+101;
    var test5 = "100"-"100";
    var test6 = "hello, "+10/2;
    var test7 = "바나나: "+3+3+"개, 사과: "+7+"개";

    var div = document.createElement("div");
    div.id = "cont";
    var result = order1+"<br>"+
                 order2+"<br>"+
                 order3+"<br><br>"+
                 test1+"<br>"+
                 test2+"<br>"+
                 test3+"<br>"+
                 test4+"<br>"+
                 test5+"<br>"+
                 test6+"<br>"+
                 test7+"<br>";
    div.innerHTML = result;
    document.body.appendChild(div);
</script>
```

```
결과
  3피자
  3피자
  12피자

  Infinity
  5454
  21 1
  200
  0
  hello, 5
  바나나: 33개, 사과: 7개
```

명시적 형변환

개발자가 코드로 직접 어떤 데이터형으로 바꿀지 명시하여 형을 변환 시키는 것을 말한다.

문자를 숫자로 형변환

변환 결과	사용 함수	예
정수 형	parseInt()	var a = "123.456"; parseInt(a); // 결과 : 123
	Number()	var a = "123"; Number(a); // 결과 : 123
실수 형	parseFloat()	var a = 123.456; parseFloat(a); // 결과 : 123.456
	Number()	var a = "123.456"; Number(a); // 결과 : 123.456

숫자를 문자로 형변환

변환 결과	사용 함수	예
일반 문자 형	String()	var a = 15; String(a); // 결과 : "15
16진수 문자 형	numObj.toString()	var a = 15; a.toString(16); // 결과 : "f"
실수(고정소수점) 문자형	numObj.toFixed()	var a = 123.456; a.toFixed(2); // 결과 : "123.46" // 반올림

9 제어문

프로그램의 실행 흐름을 제어할 때 사용하는 문장이다.

9.1 조건문

if 조건문

자바스크립트에서 가장 기본으로 쓰이는 조건문이다

- if 조건문의 기본 형태

```
if ( 조건표현식 ) {
    // 조건표현식의 결과과 참(true)일 때 수행될 문장 작성 구역
}
```

조건 표현식의 결과가 false이면 실행될 내용이 없는 구문형식이다.

- 기본적인 if 조건문 예제

```
<script>
var value1 = 100, value2 = 200;
if(value1 > value2) {     // 100 > 200 이 참일 때 실행
  alert('100 > 200 -> true');
}

alert('작업완료');
</script>
```

if ~ else 조건문

if ~ else 조건문은 거짓일 때 수행하는 코드를 나누어서 if 조건문 뒤에 else 키워드와 함께 붙여 사용한다

- if ~ else 조건문의 기본 형태

```
if ( 조건표현식 ) {
    // 조건표현식이 참일 때 수행되는 문장 작성 구역
} else {
// 위 조건표현식이 거짓일 때 수행되는 문장 작성 구역
}
```

- 기본적인 if ~ else 조건문 예제

```
<script>
var num1 = 100;
var num2 = 200;

if(num1 == num2) {      // num1 == num2 이 참일 때 실행
  alert('num1과 num2가 같다₩nnum1=' + num1 +', num2=' + num2);
} else {      // num1 == num2 이 거짓일 때 실행
  alert('num1과 num2가 같지 않다₩nnum1=' + num1 +', num2=' + num2);
}

alert('작업완료');
</script>
```

if ~ else if 조건문

if 조건문의 표현식이 거짓일 경우 무조건 수행되는 else 문과 다르게 한번 더 조건표현식을 검사할 수 있도록 만든 조건문이다.

- if else 조건문의 기본 형태

```
if ( 조건표현식1 ) {
// 조건표현식1이 참일 때 수행되는 문장
} else if ( 조건표현식2 ) {
// 조건표현식1이 거짓이고 조건표현식2가  참일 때 수행되는 문장
} else if ( 조건표현식3 ) {
// 조건표현식1,2 둘 다 거짓이고, 조건표현식3이 참일 때 수행되는 문장
} else {
// 위에 제시된 모든 조건표현식이 거짓일 때 수행되는 문장
}
```

else if 문은 여러 번 중첩 사용 가능하다 사용 횟수에는 제한이 없다.
구문 맨 마지막에 else { } 구문을 사용할 수도 있고, 사용하지 않을 수도 있다.

- 기본적인 if – else if – else 조건문

```
<script>
var num = 200;

if(num == 100) {
  // num == 100 이 참일 때 실행
  alert('num과 100과 같다');
} else if(num == 200) {
  // num == 200 이 참일 때 실행
  alert('num과 200과 같다');
} else if(num == 300) {
  // num == 300 이 참일 때 실행
  alert('num과 300과 같다');
} else {
  // num이 100, 200, 300 모두 아닐 때 실행
  alert('num은 100, 200, 300과 같지 않다');
}

alert('작업완료');
</script>
```

중첩 if 조건문

if { }과 else { } 내에 조건문을 추가적으로 작성한 것이다
조건식을 더 상세하게 구분 제어할 수 있는 장점이 있다

- 1부터 100 사이의 정수에 대한 홀수와 짝수를 구분 처리하는 프로그램 예제

```
<script>
var num = 29;

if(num >= 1 && num <=100) {
   if(num % 2 == 0) {
      alert("num은 짝수");
   } else {
```

```
    alert("num은 홀수");
    }
  } else {
    alert("num은 1-100 사이의 숫자가 아닙니다");
  }
</script>
```

switch 문

어떤 변수가 가진 값 또는 계산의 결과값에 따라 실행되는 구문을 선택해야할 때 사용하는 문장이다.

- switch 문의 기본 형태

```
switch (선택기준) {
  case 값:      문장      break;
  case 값:      문장      break;
  default:      문장
}
```

case 문은 선택에 대한 값을 제시하는 구문이며, 여러 번 사용할 수 있다 case 문 마지막에는 반드시 break 문을 사용해야 한다.

default 문은 case 구문 맨 마지막에 사용할 수 있으며, case 에서 제시된 값이 없을 경우에 작동되는 구문이다. default 문은 생략 가능하다

- 변수에 기록된 값에 대한 홀수/짝수 구분하기

```
<script>
var num = 7;

switch(num % 2) {
  case 0:  alert('짝수');  break;
  case 1:  alert('홀수');  break;
}
</script>
```

삼항연산자를 이용한 간단한 조건문

삼항연산자를 이용해 if ~ else 조건문과 같은 기능이 수행되게 할 수 있다

- 삼항연산자

```
변수 = 조건표현식 ? 참일 때 선택할 값 :거짓일 때 선택할 값
조건표현식 ? 참일 때 수행할 구문 : 거짓일 때 수행할 구문 ;
```

- 삼항연산자 예제

```
<script>
alert(true ?'참입니다' :'거짓입니다');

var num = 11;
alert(num==123 ?'num은 123과 같다' :'num은 123과 같지 않다');
alert(num<100 ?'num은 100보다 작다' :'num은 100보다 작지 않다');
</script>
```

9.2 반복문

for 반복문

반복 횟수가 정해진 반복 처리에 주로 사용되는 반목문이다.

- 기본적인 for 문

```
for( 초기식; 조건식; 증감식 ) {
    수행코드
}
```

- 수행순서
  1. 초기식을 실행
  2. 조건식과 비교하여 거짓이면 반복문 종료, 참이면
  3. 수행코드 실행
  4. 증감식 수행
  5. 증감된 값으로 다시 조건비교 처리함 (2부터 다시 반복)

- alert 5번 호출하기

```
<script>
for(var i=0; i<5; i++) {
      alert(i + '번째 반복');
}
</script>
```

변수를 이용해 반복하려는 횟수를 조절하여 코드를 반복시킬 수 있다

- 배열의 요소 출력하기

```
<script>
var arr = ['Apple', 'Banana', 'Cherry'];

for(var i=0; i<arr.length; i++) {
   alert(arr[i]);
}
</script>
```

배열의 0번째 인덱스부터 배열의 길이 (length) 만큼 반복하여 간단히 배열의 요소를 모두 출력할 수 있다

for in 반복문

배열이나 객체를 이용한 반복문을 쉽게 다룰 수 있도록 제공되는 반복문으로, 반복 카운트용 인덱스 변수값이 자동으로 0부터 시작해서 1씩 증가해서 배열명.length – 1 까지 진행하는 구문이다.

- 기본적인 for in 조건문

```
for(var i in arr) {
   수행코드
}
```

위의 코드는 다음과 같은 기본 for 반복문의 기능을 한다

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

- for in 반복문을 이용하여 배열의 요소 출력하기

```
<script>
var arr = ['Apple', 'Banana', 'Cherry'];

for(var i in arr) {
        alert(arr[i]);
}
</script>
```

- for in 반복문을 이용해서 객체정보도 접근이 가능하다.

```
<script>
    var obj = { "Korea":"Seoul",
                "US":"Washington D.C",
                "China":"Beijing",
                "France":"Paris" }

    for (var key in obj) {
        document.write(key, " : ", obj[key], "<br>");
    }
</script>
```

Korea : Seoul
US : Washingto D.C
China : Beijing
France : Paris

위와 같이 객체의 키값을 변수 key에 담으면서 순회한다. 해당 key값을 가지고, value에도 접근이 가능하다.

- 자바스크립트 배열의 forEach 메소드를 통해 배열 요소를 조회할 수 있다.

```
<script>
    var arr = ['apple','banana','kiwi'];
    arr.forEach( function(element) {
        document.write(element+"<br>")
    });
 </script>
```

while 반복문

반복횟수가 정해지지 않은 반복문에 주로 사용되며, 반복에 대한 조건식이 참일 경우 while { } 안의 구문이 실행된다.

- 기본적인 while 조건문

```
while( 반복조건식 ) {
  수행코드
}
```

- while 문을 이용한 무한 반복(실행 금지) 예

```
<script>
while(true) {    // 반복에 대한 조건식이 참(true)이면
alert(arr[i]);  // 예제 코드는 무한 반복하게 된다.
}
</script>
```

- while 반복문에 종료 조건처리 예

```
<script>
var i = 0; // 초기식
while(i<5) { // while 조건식
  alert(i);
  i++; // 증감식
}
</script>
```

while 문이 반복될 때마다 i 변수의 값이 1씩 증가되므로 무한 루프에 빠지지 않게 된다.
for문의 초기식, 조건식, 증감식을 while문으로 바꾸어서 작성하면 위의 예문과 같다.

do ~ while 반복문

while 문과 비슷하지만 do 문을 먼저 사용하고 마지막에 while(조건식) 으로 반복을 제어한다 . for 문과 while 문은 조건을 먼저 판단하여 반복을 수행할지 결정하지만, do - while문은 한번 수행한 후에 조건을 판단하여 반복을 더 수행할지 결정한다.

- 기본적인 do ~ while 조건문

```
do {
  수행코드
} while ( 조건식 );  // 마지막에 ;(세미콜론) 반드시 작성
```

- do ~ while 문을 이용한 반복문

```
<script>
var i = 0; // 초기식
do { // 무조건 실행
  alert(i);
  i++; // 증감식
} while(i<5);  // while 조건식
</script>
```

중첩 반복문

조건문을 중첩시킨 것과 마찬가지로 반복문도 중첩시켜 사용할 수 있다

- 중첩 for문을 이용한 예제

```
<script>
var output = '';

for(var i=0; i<5; i++) {
  for(var j=0; j<=i; j++) {
    output += '*';
  }
  output += '₩n';
```

```
}

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

9.3 break 문 & continue 문

break문

switch문이나 반복문을 중단시킬 때 사용한다.

- 조건문과 break 문을 섞어 무한 반복문 중단 예제

```
<script>
var i=0;
while (true) {
   alert(i + '번째 반복 수행');
   if(i == 10) {      // i가 10과 같다면 while 반복문 중단
      break;
   }
   i++;
}
</script>
```

- 반복을 수행할 때마다 확인하며 진행하는 예제

```
<script>
for(var I = 0; true; i++) {  // 조건식이 true로 무한 반복
   alert(i + '번째 반복');
   if(!confirm('계속 수행하시겠습니까?')) {  // 반복을 진행할지 확인
      break;
   }
}
alert('작업 종료');
</script>
```

* confirm() 함수는 alert()와 유사하지만 알림 창에서 OK, CANCEL 두 가지를 선택할 수 있게 된다. OK를 선택하면 결과로 true를 반환하며, CANCEL을 선택하면 false를 반환한다.

continue 문

반복 실행 내용의 중간 생략을 원할 때 사용하는 구문이다..

while문은 조건식을 검사하게 되며, for문은 증감식을 수행한다.

- continue문 사용 예

```
<script>
for(var I = 0; I < 10; i++) {
  continue;
  alert(i); // 수행되지 않음
}
alert('작업 완료');
</script>
```

10 함수

함수란 특정 기능을 구현한 코드를 독립된 단위로 만들어 재사용하고자 할 때 사용하는 문법이다. 함수를 사용하여 코드를 작성하면 유지보수가 간편해지고 중복 코드를 줄일 수 있으며 코드 재사용성을 증대시킬 수 있다. 또한 가독성이 좋아진다.

10.1 함수 형태 1 – 기본 함수 형태

함수 외부에서 함수 내부로 전달되는 매개변수가 없고 함수 내부의 정보를 전달하는 리턴값도 없는 구조

```
<script>
function 함수이름( ) {
  실행 구문;
  …
}
</script>
```

함수를 정의하기 위해서는 function 키워드 다음에 함수 이름을 정의한다. 함수의 이름은 유일해야 하고 만들려는 함수의 기능을 함축한 의미를 가지도록 하는 것이 좋다. 함수의 기능을 { } 안에 구현한다. 정의된 함수를 호출하기 위해서는 "함수이름( )" 형식으로 사용한다

10.2 함수 형태 2 – 매개변수가 있는 함수

기본 함수 형태에서 매개변수가 추가된 형태이다.

```
<script>
function 함수이름 (매개변수1, 매개변수2,  … ) {
  실행 구문;
  …
}
</script>
```

매개변수는 함수 외부에서 함수를 호출하며 전달해야 하는 값이 있을 때 사용한다. 함수 외부에서 함수 내부로 값을 전달하는 매개체 역할을 수행하며 함수 이름 오른쪽에 ( ) 를 이용하여 필요한 매개변수를 지정하여 정의한다. 매개변수의 개수에는 제한이 없다.

매개변수가 있는 함수를 호출하기 위해서는 함수에 정의된 매개변수의 개수에 맞게 전달값을 넣어 주어야 한다. 호출방식은 "함수이름(전달값1, 전달값2, …)" 형식으로 사용한다

10.3 함수 형태 3 – 리턴값이 있는 함수

리본 함수 형태에서 리턴값이 추가된 형태이다.

```
<script>
function 함수이름( ) {
  실행 구문;
  …
  return 리턴값;
}
</script>
```

리턴값은 함수가 종료되는 시점에 함수를 호출한 함수 외부에 결과를 전달하는 역할을 한다. 매개변수는 함수가 실행될 때 값을 전달받는 역할을 한다면, 리턴값은 함수가 종료될 때 결과값을 반환해주는 역할은 한다. return 값; 으로 표현하며, 소스 코드 중간에 값 없이 함수 실행을 중단시키고자 할 경우에도 if 조건문과 함께 return; 을 사용할 수도 있다.

11. 변수와 함수의 관계

자바스크립트의 변수는 데이터 타입을 명시하지 않고 만들 수 있으며, 사용할 때에도 데이터 타입을 명시하여 사용하지 않는다. 자바스크립트에서는 변수에 문자형, 숫자형, 배열을 담아 사용할 수 있지만 함수 또한 변수에 담아 사용할 수 있다.

11.1 전역변수와 지역변수

var 키워드를 사용해 스크립트 어디서나 변수를 선언할 수 있다. 하지만, 변수를 어디에서 선언했는냐에 따라, 이 변수를 어디에서 사용할 수 있는지 변수의 사용영역(scope)이 제한된다.

전역변수 (Global Variables)

함수 밖에서 정의한 변수는 전역범위 Global Scope를 갖게 된다. 페이지가 로딩될 때 생성되어 페이지를 닫기전까지 살아 있다. 페이지를 리로드하면, 기존 전역변수는 사라지고, 새로운 전역변수가 생긴다.

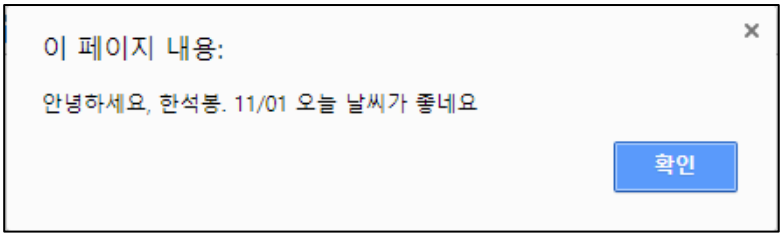
지역변수 (Local Variables)

함수 안에서 정의한 변수는 지역범위 Local Scope를 갖게 된다. 함수가 호출될 때 생성되고 함수가 반환되면 사라진다. 함수의 매개변수 역시 지역변수에 해당되며, 반환값으로 전달되지 않는 한 모두 사라진다.

```
<script>
    var gv_date = "11/01";
    var gv_str = "저는 전역변수 입니다.";
    var gv_name = "한석봉";

    helloJS(gv_name);
    function helloJS(name){
        var weather = "좋네요";
        alert("안녕하세요, "+name+". "+gv_date+" 오늘 날씨가 "+weather);
    }

    console.log(gv_name);
    console.log(weather);
</script>
```



gv_로 시작되는 3개의 변수는 전역변수이다. 어디서나 접근 가능하며 페이지 종료시까지 사용 가능하다. 마지막 콘솔 결과에도 gv_name 은 정상적으로 출력되는 것을 볼 수 있다.
함수 helloJS의 파라미터 name과 weather은 지역변수이다. 함수가 호출되고 반환되며 이 변수들은 사라진다. 마지막 콘솔결과에도 weather는 undefined 라면서 참조 에러를 유발하고 있다.

```
<script>
    var gv_date = "11/01";
    var gv_str = "저는 전역변수 입니다.";
    var gv_name = "한석봉";

    helloJS(gv_name);
    function helloJS(name){
        weather = "좋네요";
        alert("안녕하세요, "+name+". "+gv_date+" 오늘 날씨가 "+weather);
    }

    console.log(gv_name);
    console.log(weather);
</script>
```

결과
한석봉
좋네요

전역변수가 나중에 어플리케이션의 악영향을 미칠 수도 있으므로, 지역변수는 var 키워드와 함께 선언하는 습관을 기르자.

함수를 변수에 담으면 변수명은 함수명과 동일하게 동작한다

```
<script>
function hello() {
  alert("Hello JavaScript");
}
var func = hello;
func();
</script>
```

변수명을 함수명과 동일하게 사용할 수 있으므로 함수의 매개변수에 함수명을 전달하여 사용하는 것도 가능하다

```
<script>
function hello1() {
  alert("Hello JavaScript");
}
function hello2() {
  alert("안녕하세요");
}
function exam(func) {
  func();
}
exam(hello1);
exam(hello2);
</script>
```

매개변수 뿐만 아니라 함수의 리턴값으로 함수명을 전달하는 것도 가능하다

```
<script>
function makeHello() {
  function hello(name) {
    document.write("안녕하세요, " + name);
  }
  return hello;
```

```
}
var result = makeHello();
result("홍길동");
</script>
```

```
>> 실행결과
안녕하세요, 홍길동
```

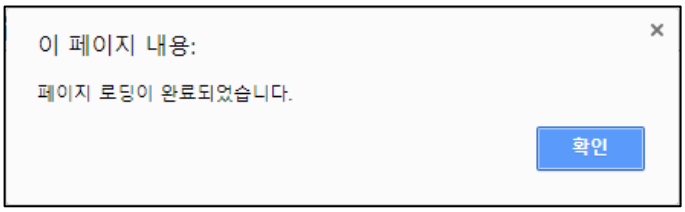
12. 함수의 분류

자바스크립트의 함수를 크게 2가지로 분류할 수 있다.
사용자 정의 함수와 자바스크립트 코어 함수이다.
사용자 정의 함수는 개발자가 직접 기능을 넣어 만든 함수를 말하며, 자바스크립트 코어 함수는 parseInt(), parseFloat() 과 같은 미리 만들어져서 자바스크립트에서 제공하는 함수를 말한다.
자바스크립트 함수를 사용 방법에 따라 분류하면 일반 함수, 중첩 함수, 콜백 함수로 분류할 수 있다.

12.1 이벤트핸들러

이벤트가 발생했을 때 처리로직을 담은 함수를 정의하고 등록해 보자.
페이지의 로딩이 끝나면 실행되는 함수를 선언하고, window.onload 속성에 할당해 보자.

```
<script>
    window.onload = pageLoadedHandler;
    function pageLoadedHandler(){
        alert("페이지 로딩이 완료되었습니다.")
    }
</script>
```



또, 아래와 같이 addEventListner 함수를 사용해서, 이벤트 이름과 이벤트 핸들러를 인자로 전달할 수 도 있다. 속성명과 달리 이벤트명은 load 이다.

```
<script>
window.addEventListener("load", pageLoadedHandler);
function pageLoadedHandler(){
        alert("페이지 로딩이 완료되었습니다.")
    }
</script>
```

또는 아래와 같이 body 태그의 onload 속성에서 함수를 직접 호출하는 인라인 방식으로 이벤트 핸들러를 연결할 수도 있다.

```
<body onload="pageLoadedHandler();">
<script>
function pageLoadedHandler(){
        alert("페이지 로딩이 완료되었습니다.")
}
</script>
</body>
```

버튼 엘리먼트를 추가하고, 클릭했을 때 이벤트핸들러를 추가해 보자.

```
<body>
    <button id="myButton">버튼</button>
<script>
    var btn = document.querySelector("button");
    var btnClickHandler = function€{
        var target = e.target;
        alert(target.id + "버튼을 클릭하셨습니다.");
    }
    btn.onclick = btnClickHandler;
</script>
</body>
```

버튼 엘리먼트도 addEventHandler 함수를 사용할 수 있다. 속성명과 달리 이벤트명은 click이다.

```
Var btn = document.querySelector("button");
btn.addEventHandler('click',btnClickHandler);
```

또, button의 onclick 속성에 해당 함수를 직접 호출하는 인라인 방식을 사용해 보자.

```
<button id="myButton" onclick="btnClickHandler();">버튼</button>
```

이벤트핸들러 함수는 이벤트 객체를 파라미터로 받을 수 있다. 이 이벤트 객체에는 이벤트 처리에 유용한 정보를 담고 있다.

속성	설명
type	이벤트타입 – click, load
target	이벤트를 발생시킨 객체에 대한 참조. 보통은 엘레먼트객체
timestamp	이벤트발생 시각정보
keyCode	사용자가 누른 키정보
clientX	이벤트발생 click 위치. 윈도우좌측으로부터의 px거리값
clientY	이벤트발생 click 위치. 윈도우위쪽으로부터의 px거리값
touches	터치디바이스에서 몇 개의 터치가 발생했는지 여부.

버튼 클릭 이벤트 핸들러에 인자로 받은 이벤트 객체의 정보를 열람하는 코드를 추가하고 직접 확인해 보자.

```
<body>
    <button id="myButton">버튼</button>
<script>
    var btn = document.querySelector("button");
    var btnClickHandler = function€{
        var target = e.target;
        alert(target.id + "버튼을 클릭하셨습니다.");

        //이벤트객체 조회
        for (var v in e) {
            console.log(v + " : "+e[v])
```

```
        }
    }
    btn.onclick = btnClickHandler;
</script>
</body>
```

```
결과
isTrusted : true
- 54 -imest : 483
- 54 -imest : 138
- 54 -imest : 44
- 54 -imest : 22
ctrlKey : false
shiftKey : false
altKey : false
metaKey : false
button : 0
buttons : 0
relatedTarget : null
pageX : 44
pageY : 22
x : 44
y : 22
- 54 -imest : 34
- 54 -imest : 12
- 54 -imestam : 0
- 54 -imestam : 0
fromElement : null
toElement : [object HTMLButtonElement]
layerX : 44
layerY : 22
getModifierState : function getModifierState() { [native code] }
initMouseEvent : function initMouseEvent() { [native code] }
view : [object Window]
```

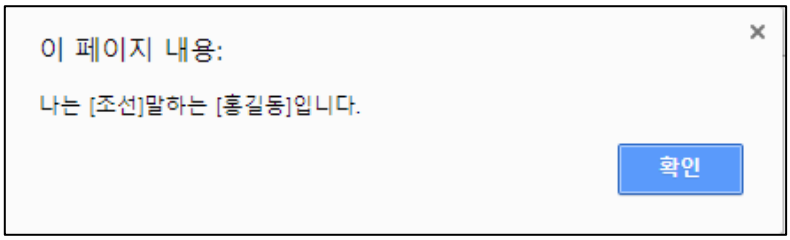
```
detail : 1
sourceCapabilities : [object InputDeviceCapabilities]
which : 1
initUIEvent : function initUIEvent() { [native code] }
NONE : 0
CAPTURING_PHASE : 1
AT_TARGET : 2
BUBBLING_PHASE : 3
type : click
target : [object HTMLButtonElement]
currentTarget : [object HTMLButtonElement]
eventPhase : 2
bubbles : true
cancelable : true
defaultPrevented : false
composed : true
- 55 -imestamp : 1947.9100000000003
srcElement : [object HTMLButtonElement]
returnValue : true
cancelBubble : false
path : [object HTMLButtonElement],[object HTMLBodyElement],[object HTMLHtmlElement],[object HTMLDocument],[object Window]
composedPath : function composedPath() { [native code] }
stopPropagation : function stopPropagation() { [native code] }
stopImmediatePropagation : function stopImmediatePropagation() { [native code] }
preventDefault : function preventDefault() { [native code] }
initEvent : function initEvent() { [native code] }
```

12.2 사용자정의 객체

객체는 리터럴로 직접 속성과 메소드를 지정해 만들 수 있다. 하지만, 동일한 구조의 객체를 여러 개 생성해야 하는 경우는 생성자 함수를 이용할 수 있다.

```
<script>
    var hong = {
```

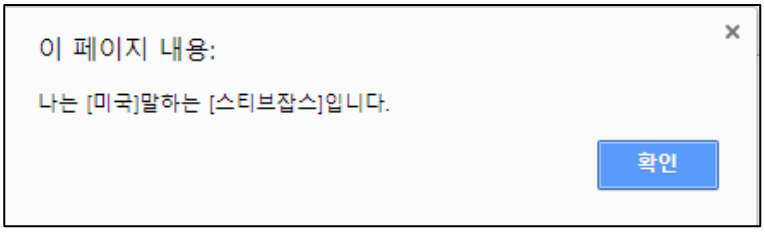
```
        name : "홍길동",
        age : 567,
        gender : "M",
        nationality : "조선",
        speak : function(){
            alert("나는 ["+this.nationality+"]말하는 ["+this.name+"]입니다.");
        }
    }
    hong.speak();
</script>
```



생성자 함수명은 관례에 따라 대문자로 시작하고, 객체 정보를 담고 있다. 자바에서 새로 객체를 생성하기 위해 해당 클래스의 생성자를 호출한 것과 닮아 있지만, 자바스크립트는 클래스 베이스의 객체지향 언어는 아니다.

```
<script>
    var steve = new Person("스티브잡스", 62, "M", "미국");
    steve.speak();

    function Person(name, age, gender, nationality){
        this.name = name;
        this.age = age;
        this.gender = gender;
        this.nationality = nationality;
        this.speak = function(){
            alert("나는 ["+this.nationality+"]말하는 ["+this.name+"]입니다.");
        }
    }
</script>
```



객체지향 언어인 자바스크립트에서 상속은 어떻게 구현하는지 알아보자.
여기 개발자 Developer 집단이 있다. 이들은 Person 에 해당하는 속성과 메소드를 모두 가지면서 직군, 경력 등의 추가 속성과 program 이라는 메소드 또한 가져야 한다.
자바스크립트에서는 프로토 타입을 통해 객체 상속을 지원한다

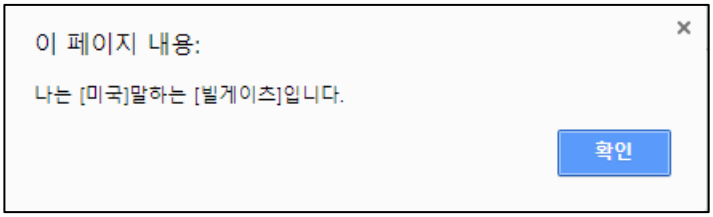
```
<script>
    var steve = new Person("스티브잡스", 62, "M", "미국");
    steve.speak();

    function Person(name, age, gender, nationality){
        this.name = name;
        this.age = age;
        this.gender = gender;
        this.nationality = nationality;
        this.speak = function(){
            alert("나는 ["+this.nationality+"]말하는 ["+this.name+"]입니다.");
        }
    }

    //Developer생성자함수 선언
    function Developer(name, age, gender, nationality, job, exp){
        this.name = name;
        this.age = age;
        this.gender = gender;
        this.nationality = nationality;
        this.job = job;
        this.exp = exp;
        this.work = function(){
```

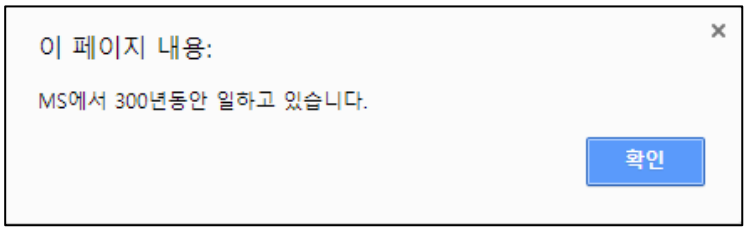
```
            return job+"에서 "+exp+"년동안 일하고 있습니다.";
        }
    }
    //Person을 prototype으로 상속한다.
    Developer.prototype = new Person();

    var bill = new Developer("빌게이츠", 62, "M", "미국", "MS", 300);
    bill.speak();
    alert(bill.work());
</script>
```



Developer의 프로토타입으로 매개변수 없는 Person 생성자를 전달하면서 Person 객체를 상속받고 있다. 직접 선언하지 않은 부모 객체의 속성에도 접근할 수 있다. 또한 Developer의 생성자 함수에서 부모 객체 Person 함수의 call 메소드를 사용할 수도 있다.

```
<script>
function Developer(name, age, gender, nationality, job, exp){
        //부모객체 생성자호출
        Person.call(this, name, age, gender, nationality);
        this.job = job;
        this.exp = exp;
        this.work = function(){
            return job+"에서 "+exp+"년동안 일하고 있습니다.";
        }
    }
</script>
```



12.3 JSON 객체

자바스크립트는 웹에서의 프로그래밍 언어일뿐 아니라, 객체를 저장하고 전송하는 공통교환 포맷이 되어 가고 있다. JSON은 Javascript Object Notation의 약자로 자바스크립트 객체를 문자열로 표현할 수 있는 포맷이다.
앞선 코드에 이를 추가해 보자.

```
<script>
    var strBill = JSON.stringify(bill);
    console.log(strBill);
    var billAgain = JSON.parse(strBill);
    console.log(billAgain);
</script>
```

```
JSON객체_문자열변환결과 :
{"name":"빌게이츠","age":62,"gender":"M","nationality":"미국","job":"MS","exp":300}

자바스크립트 객체로 재변환결과 :
[object Object]
```

위와 같이 JSON포맷으로 변환한 문자열은 메소드는 자동 삭제처리 되지만, 하나의 문자열이므로 객체, 배열은 물론이고, 모든 기본형과 함께 사용할 수 있다.

12.4 중첩 함수

함수 내부에 함수 구문을 추가한 것

```
<script>
function outer() {
  function inner() {
  }
  innter();
}
</script>
```

함수 내부에 중첩시킨 함수는 하나 이상 만들 수 있다. 함수 내부에서만 사용하는 기능을 그룹화 하기 위해 많이 사용한다. 중복 코드를 줄이는 장점이 있으며 그룹화된 코드를 관리하기가 용이해진다. 중첩된 함수는 외부 함수의 지역변수를 사용할 수도 있다.

12.5 콜백 함수

함수 내부의 처리결과를 담당할 외부 함수를 지정할 때 사용

return 문과 비슷하지만 return문은 결과값만을 전달한다면 콜백 함수는 결과값을 처리하는 방법까지 지정하는 것이다

콜백 함수의 구조는 로직 구현 부분과 로직 처리 부분으로 나누어 작성한다. 로직 구현은 동일하고 로직에 대한 결과를 처리하는 부분이 다양할 때 사용한다.

- 전달된 매개변수 2개의 덧셈 결과를 두 가지 방식으로 처리

```
<script>
function add(num1, num2, callback) {
  var result;
  result = num1 + num2;
  callback(result);
}
function print1(result) {
  alert("두 수의 합은 " + result + "입니다");
}
function print2(result) {
  document.write("결과는 " + result + "입니다");
```

```
}

add(10, 20, print1);
add(10, 20, print2);
</script>
```

callback이라는 이름의 매개변수를 지정하였고 callback을 이용하여 로직의 결과를 처리하는 함수를 호출하도록 구현하였다. callback 매개변수의 이름은 반드시 callback일 필요는 없다. add() 함수를 호출할 때 alert로 결과를 처리하는 방식의 print1함수를 지정하거나 document.write로 결과를 처리하는 print2함수를 지정함에 따라 add함수의 결과 처리 방식이 달라지게 된다. 이 코드는 return구문을 이용하여 구현하는 것도 가능하다

- 위의 콜백함수를 return 구문으로 구현

```
<script>
function add(num1, num2, callback) {
  var result;
  result = num1 + num2;

  return result;
}

var result = add(10, 20);
alert("두 수의 합은 " + result+ "입니다");
document.write("결과는 " + result + "입니다");
</script>
```

13. 자바스크립트 라이브러리

13.1 자바스크립트 내장함수 및 객체

① 타이머 함수

일정 시간마다 특정 구문을 실행하고자 할 때 사용하는 함수이다.

주요 기능

타이머 함수는 전역 객체인 window 객체에 들어 있다

window.setInterval() 처럼 사용할 수 있으며 setInterval() 처럼 사용할 수도 있다

함수	설명
setInterval()	일정 시간마다 주기적으로 특정 구문을 실행하는 기능
setTimeout()	일정 시간이 지난 후 특정 구문을 한번 실행하는 함수
clearInterval()	실행 중인 타이머를 멈추는 기능

일정 시간마다 주기적인 실행 – setInterval()

```
var timerID = setInterval(func, duration);
```

func : 지연 시간마다 타이머 함수에 의해 호출되는 함수

duration : 지연 시간, 단위는 밀리초

함수 호출시 고유한 타이머 ID가 리턴된다. 이는 clearInterval()함수 호출시 사용할 수 있다.

1000밀리초(5초) 마다 알림창을 띄우는 예제

```
<script>
function hello() {
  alert("Hello");
}
setInterval(hello, 5000);
</script>
```

hello() 함수를 익명 함수로 작성한 예제

```
<script>
setInterval(function() {
  alert("Hello");
}, 1000);</script>
```

일정 시간 후 한번만 실행 – setTimeout()

```
var timerID = setTimeout(func, duration);
```

func : 지연 시간이 지난 후 타이머 함수에 의해 호출되는 함수

duration : 지연 시간, 단위는 밀리초

함수 호출시 고유한 타이머 ID가 리턴된다. 이는 clearTimeout()함수 호출시 사용할 수 있다.

2000밀리초(1초) 후 알림창을 띄우는 예제

```
<script>
setTimeout(function() {
  alert("Hello");
}, 1000);
</script>
```

타이머 멈추기 – clearInterval(timerID);

```
clearInterval(timerID);
```

timerID : 제거할 타이머 ID

1초 마다 증가하는 숫자를 출력하고, 숫자가 5가 되면 타이머를 종료하는 예제

```
<script>
var count = 0;
var tID = setInterval(function () {
   count++;
   alert(count);

   if(count == 5) {
      clearInterval(tID);
   }
}, 1000);
</script>
```

② Math 클래스

수학계산과 관련된 유용한 기능이 담겨있는 클래스로써, Math 클래스의 기능은 인스턴스 생성없이 즉시 사용할 수 있다

숫자를 랜덤하게 생성하는 기능, 사인(sin) 및 코사인(cos) 과 같은 수학 관련 기능이 담겨 있다

주요 기능

주요 프로퍼티

프로퍼티	설명
PI	원주율 값

주요 메소드

메소드	설명
abs()	절대값 반환
acos()	아크코사인 값 반환
asin()	아크사인 값 반환
atan()	아크탄젠트 값 반환
ceil()	올림 값 반환
cos()	코사인 값 반환
floor()	내림값 반환
log()	자연로그 값 반환
max()	두 수 중 큰 값 반환
min()	두 수 중 작은 값 반환
random()	0과 1 사이의 난수 값 반환
round()	가장 가까운 정수로 반올림한 값 반환
sin()	사인 값 반환
sqrt()	제곱근 반환
tan()	탄젠트 값 반환

랜덤 숫자 만들기 – Math.random()

random() 함수는 0과 1 사이의 실수 형태의 숫자를 반환한다.

```
<script>
var value = Math.random();
</script>
```

원하는 범위의 난수 얻기

Math.random()을 통해 얻어진 실수에 원하는 총 경우의 수를 곱하고, 최소값을 더한다.

경우의 수는 최대값-최소값으로, 최소값이 포함되고, 최대값은 포함되지 않는다.

```
<script>
var value = Math.random() * (경우의 수)+최소값
</script>
```

1초마다 50~100 사이의 정수 출력하는 예제

```
<script>
setInterval(function() {
   alert(parseInt(Math.random() * 50) + 50);
}, 1000);
</script>
```

작은 값, 큰 값 알아내기 – Math.min(), Math.max()

Math.min() 함수는 인수로 들어온 두 수 중 작은 값을 반환한다

Math.max() 함수는 인수로 들어온 두 수 중 큰 값을 반환한다

```
<script>
var min = Math.min(num1, num2);
var max = Math.max(num1, num2);
</script>
```

올림값, 내림값, 반올림값 구하기 – Math.floor(), Math.ceil()

Math.abs() 절대값을 반환하는 함수

```
<script>
      Math.abs('-1');       // 1
      Math.abs(-2);         // 2
      Math.abs(-0.43);      // 0.43
      Math.abs(null);       // 0
      Math.abs('');         // 0
      Math.abs([]);         // 0
      Math.abs([2]);        // 2
      Math.abs([1,2]);      // NaN
      Math.abs({});         // NaN
      Math.abs('string');   // NaN
      Math.abs();           // NaN
</script>
```

Math.ceil() 올림값 반환함수

```
<script>
      Math.ceil(.95);      // 1
     Math.ceil(4);         // 4
     Math.ceil(7.004);   // 8
     Math.ceil(-0.95);   // -0
     Math.ceil(-4);       // -4
     Math.ceil(-7.004); // -7
</script>
```

15

소수점 두번째자리까지 올림수 만들기. 소수점 이하에서 처리되기 때문에, 원하는 자리수만큼 곱했다가, ceil 함수 처리후 다시 나누기해 준다.

```
<script>
  var num = 345.678;
  num = Math.ceil(num*100)/100;
 console.log(num);//345.68
</script>
```

Math.floor() 버림값 반환함수

```
<script>
      Math.floor( 45.95); //   45
      Math.floor( 45.05); //   45
      Math.floor(   4    ); //    4
      Math.floor(-45.05); // -46
      Math.floor(-45.95); // -46
</script>
```

소수점 두번째 자리까지 내림수 만들기. 소수점 이하에서 처리되기 때문에, 원하는 자리수만큼 곱했다가, floor 함수처리 후 다시 나누기해 준다.

```
<script>
  var num = 345.678;
  num = Math.floor(num*100)/100;
 console.log(num);//345.68   </script>
```

Math.rount() 반올림값 반환함수

```
<script>
Math.round( 20.49); //  20
Math.round( 20.5);  //  21
Math.round( 42   );  //  42
Math.round(-20.5);  // -20
Math.round(-20.51); // -21
</script>
```

소수점 두번째 자리까지 반올림수 만들기. 소수점 이하에서 처리되기 때문에, 원하는 자리수만큼 곱했다가, round 함수처리 후 다시 나누기해 준다.

```
<script>
  var num = 345.678;
  num = Math.round(num*100)/100;
 console.log(num);//345.67
</script>
```

③ String 클래스

문자열을 생성하거나 문자열의 길이를 알아내는 등의 문자열을 다루는 기능을 가지고 있는 클래스

주요 기능

주요 프로퍼티

프로퍼티	설명
length	문자열 길이

주요 메소드

메소드	설명
indexOf(str)	str 문자열이 위치한 인덱스 구하기, 앞에서부터 검색
lastIndexOf(str)	str 문자열이 위치한 인덱스 구하기, 뒤에서부터 검색
match(reg)	정규표현식을 활용한 문자열 검색
replace(reg, reg)	정규표현식을 활용한 문자열 교체
search(reg)	정규표현식을 활용한 문자열 위치 검색
slice(start, end)	start번째부터 end번째 문자열 검색

split(str)	문자열을 str로 분할해 배열로 생성
substring(startIndex, endIndex)	start 번째부터 문자열 추출
substr(start, length)	start 번째부터 count개수만큼 문자열 추출
toLowerCase()	모든 문자열을 소문자로 변환
toUpperCase()	모든 문자열을 대문자로 변환
trim()	좌우 공백 제거

문자열 생성하기

문자열을 생성하는 방법으로는 리터럴(문자열 상수) 방식으로 만드는 방법과 String 객체를 생성해 만드는 방법이 있다

자바스크립트는 문자열을 리터럴 방식으로 만들어 사용하더라도 내부적으로 String 클래스의 인스턴스를 생성하게 된다. 따라서 리터럴 방식으로 생성된 문자열도 String 클래스의 메소드와 프로퍼티를 이용할 수 있다.

리터럴 방식

```
<script>
var str = "Hello";
</script>
```

String 객체 생성 방식

```
<script>
var str = new String("Hello");
</script>
```

문자열 길이 알아내기 – length 프로퍼티

문자열이 생성되면 length 프로퍼티를 이용해 문자열의 길이를 알 수 있다

```
<script>
document.write("Hello".length); // 5
</script>
```

```
<script>
var str = "Hi";
alert(str.length); // 2
</script>
```

```
<script>
var str = new String("Hello World");
alert(str.length); // 11
</script>
```

문자열의 특정문자 index 구하기 – indexOf()

해당문자가 존재하면 해당인덱스값을 리턴하고, 그렇지 않다면, -1을 리턴한다.

```
<script>
    'khacademy_kh정보교육원'.indexOf('kh');       // returns  0
    'khacademy_kh정보교육원'.indexOf('khh');      // returns -1
    'khacademy_kh정보교육원'.indexOf('kh', 0); // returns  0
    'khacademy_kh정보교육원'.indexOf('kh', 5); // returns  10
    'khacademy_kh정보교육원'.indexOf('에이치', 7); // returns -1
    'khacademy_kh정보교육원'.indexOf('');             // returns  0
    'khacademy_kh정보교육원'.indexOf('', 9);         // returns  9
    'khacademy_kh정보교육원'.indexOf('', 10);       // returns 10
    'khacademy_kh정보교육원'.indexOf('', 18);       // returns 10
</script>
```

문자열의 특정문자 뒤에서부터 찾아 index 값을 구하기 – lastIndexOf()

두번째 인자값은 검색을 시작할 인덱스이다. lastIndexOf('a', 2)라고 하면, a라는 문자를 해당문자열 인덱스 2, 1, 0 의 순서로 검색해서 존재하면, 해당인덱스값을, 그렇지 않다면 -1을 리턴한다.

```
<script>
    'canal'.lastIndexOf('a');       // returns 3
    'canal'.lastIndexOf('a', 2);   // returns 1
    'canal'.lastIndexOf('a', 0);   // returns -1
    'canal'.lastIndexOf('x');       // returns -1
    'canal'.lastIndexOf('c', -5); // returns 0
    'canal'.lastIndexOf('c', 0);   // returns 0
    'canal'.lastIndexOf('');        // returns 5
    'canal'.lastIndexOf('', 2);    // returns 2
</script>
```

특정 위치 문자 구하기 – charAt()

```
<script>
var ch = 문자열.charAt(index);
</script>
```

index : 0부터 시작하여 구하려는 위치를 지정하는 인덱스 값
index위치의 문자를 반환한다

```
<script>
var alpha = "ABCDEFG";
alert(alpha.charAt(3)); // D
</script>
```

특정 문자열을 원하는 문자열롤 변경하기. – replace()

첫번째 인자로, 검색할 문자열 또는 정규식을 전달하고, 두번째 인자로 교체할 값이나, 함수를 전달한다. 자바스크립트 String.replace()는 검색된 문자열 하나만 교체하므로, 여러 건의 교체를 원하는 경우 정규식을 사용해야 한다. 아래 코드를 실행해 보자.

```
Replace(regExp|substr, newSubstrr|function)
<script>
    var str = 'khacademy_KHACADEMY';
     var newstr = str.replace('academy', '정보교육원');
     console.log(newstr);
</script>
kh정보교육원_KHACADEMY
```

위와 같이 처음 만난 academy만 정보교육원으로 변경되었다. 정규식을 사용해보자. 정규식 옵션으로 문자열 전체를 의미하는 g, 대소문자를 구분하지 않는 i를 추가한다.

```
<script>
    var newstr = str.replace(/academy/gi, '정보교육원');
        console.log(newstr);
</script>
kh정보교육원_KH정보교육원
```

문자열을 지정한 구분자로 쪼개어 배열로 돌려받기 – split()

splitString 함수를 선언하고 각각의 문자열을 인자로 호출해 보자.

```
<script>
        function splitStrig(str, separator){
            var strArr = str.split(separator);
            return strArr;
        }

        var str1 = "kh_케이에이치_KH";
        var str2 = "2016,2017,2018,2019";
        var str3 = "안녕, 나 자바스크립트야!"

        strArr1 = splitStrig(str1, "_");
        strArr2 = splitStrig(str2, ",");
        strArr3 = splitStrig(str3, "");

        console.log("["+strArr1+"]");
        console.log("["+strArr2+"]");
    console.log("["+strArr3+"]");</script>
```

```
[kh,케이에이치,KH]
[2016,2017,2018,2019]
[안,녕,,, ,나, ,자,바,스,크,립,트,야,!]
```

문자열 추출하는 함수 – substring(), substr()

- substring(startIndex, endIndex)I – 첫번째인자 인덱스값 문자열부터 두번째인자 인덱스값 전까지 추출한다. 두번째 인자를 생략하면, 문자열 끝까지 가져온다.

```
Str.substring(indexStart[, indexEnd]);
```

```
<script>
    var namestr = "홍길동신사임당유관순";
    var name1 = namestr.substring(0,3);
    var name2 = namestr.substring(3,7);
    var name3 = namestr.substring(7);
```

```
        console.log(name1);
        console.log(name2);
        console.log(name3);
    </script>
```

```
홍길동
신사임당
유관순
```

- substr(startIndex, length) – 첫번째 인자 문자열부터 length 길이만큼 문자열을 추출. 두번째 인자를 생략하면, 문자열 끝까지 가져온다.

```
    Str.substr(start[, length])
```

```
<script>
    var namestr = "홍길동신사임당유관순";
    var name1 = namestr.substr(0,3);
    var name2 = namestr.substr(3,4);
    var name3 = namestr.substr(7);

    console.log(name1);
    console.log(name2);
    console.log(name3);
</script>
```

```
홍길동
신사임당
유관순
```

좌우공백 제거함수 – trim()

제거한후에는 새로운 문자열을 리턴한다. 기존 문자열은 변경하지 않는다

```
<script>
    var strtrim = "        white    Space        "
    var newstr = strtrim.trim();
   console.log(newstr);
</script>
```

```
white    Space
```

④ Date 클래스

날짜 및 시간과 관련된 기능이 담겨있는 클래스

주요 메소드

메소드	설명
getDate()	로컬시간을 사용하여 일을 반환
getDay()	로컬시간을 사용하여 요일을 반환
getFullYear()	로컬시간을 사용하여 년도를 반환
getHours()	로컬시간을 사용하여 시간을 반환
getMinutes()	로컬시간을 사용하여 분을 반환
getMilliseconds()	로컬시간을 사용하여 밀리초를 반환
getMonth()	로컬시간을 사용하여 월을 반환
getSeconds()	로컬시간을 사용하여 초를 반환
getTime()	1970년 1월 1일 00시 00분 00초를 기준으로 현재까지 경과한 시간을 밀리초로 반환
getYear()	로컬시간을 사용하여 년도를 반환. getFullYear()를 사용하는 것이 좋다

시간, 분, 초, 밀리초 알아내기 – getHours(), getMinutes(), getSeconds(), getMilliseconds()

```
<script>
var d = new Date();
var hours = d.getHours();
</script>
```

0-23의 정수로 시간을 반환한다

```
<script>
var d = new Date();
var minutes = d.getMinutes();  //0-59의 정수로 분을 반환한다
</script>
```

```
<script>
var d = new Date();
var seconds = d.getSeconds();  //0-59의 정수로 초를 반환한다
</script>
```

```
<script>
var d = new Date();
var mSeconds = d.getMilliseconds();
</script>
```

0-999의 정수로 밀리초를 반환한다

현재 시간 출력하기 예제

```
<script>
var d = new Date();
alert(d.getHours() + "시 " + d.getMinutes() + "분 " + d.getSeconds() + "초 ");
</script>
```

년, 월, 일, 요일 알아내기 – getFullYear(), getMonth(), getDate(), getDay()

```
<script>
var d = new Date();
var year = d.getFullYear();
</script>
```

네 자리 숫자의 년도 반환

```
<script>
var d = new Date();
var month = d.getMonth()+1;
</script>
```

0-11의 정수, 즉 인덱스값으로 반환하기 때문에 1을 더해준다.

```
<script>
var d = new Date();
var date = d.getDate();    //1-31의 정수로 날짜 반환
</script>
```

```
<script>
var d = new Date();
var day = d.getDay();    //0(일)-6(토)의 정수로 요일 반환
</script>
```

현재 년월일, 요일 출력

```
<script>
var d = new Date();
var date = d.getFullYear() + "/" + (d.getMonth()+1) + "/" + d.getDate() + ", ";
switch(d.getDay()){
case 0:
   date += "일";
   break;
case 1:
   date += "월";
   break;
case 2:
   date += "화";
   break;
case 3:
   date += "수";
   break;
case 4:
   date += "목";
   break;
case 5:
   date += "금";
   break;
case 6:
   date += "토";
   break;
}

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

월은 0-11이므로 +1 해주어야 한다

요일은 0-6의 숫자를 반환하므로 요일에 해당하는 문자로 변경해줄 필요가 있다

화면에 현재날짜, 시각을 실시간 반영하는 코드를 구현해보자.

```
<script>
  var func = function(){
      var d = new Date();
      var year = d.getFullYear();
      var month = d.getMonth()+1;
      var date   = d.getDate();
      var day = d.getDay();
      var hour = d.getHours();
      var min = d.getMinutes();
      var sec = d.getSeconds();
      var milisec = d.getMilliseconds();

      month = (month>9?"":"0")+month;
      date = (date>9?"":"0")+date;
      hour = (hour>9?"":"0")+hour;
      min = (min>9?"":"0")+min;
      sec = (sec>9?"":"0")+sec;
      var datestr;
      switch(day){
          case 0: datestr = "일"; break;
          case 1: datestr = "월"; break;
          case 2: datestr = "화"; break;
          case 3: datestr = "수"; break;
          case 4: datestr = "목"; break;
          case 5: datestr = "금"; break;
          case 6: datestr = "토"; break;
      }
      return year+"-"+month+"-"+date+" ("+datestr+") "+hour+":"+min+":"+sec+":"+milisec;
  }
  setInterval(function(){
      document.body.innerHTML = func();
  },10);
</script>
```

yyyy-mm-dd (ddd) hh:mm:ss:milsec 의 포맷 중에서 현재 달, 날짜, 시각, 분, 초 등의 포맷을 두 자리 수로 맞추기 위한 코드는 각각 3항 연산자로 구현하였다. 요일은 0~6 까지의 인덱스 값을 실제 요일 데이터로 변환하기 위해 switch 문을 사용하였고, setInterval 함수를 사용해서 10 밀리세컨드 마다 날짜를 갱신한다.

⑤ Array 클래스

배열을 리터럴 방식([데이터1, 데이터2, …])으로 사용하더라도 String과 마찬가지로 인스턴스를 자동으로 생성해 변수에 담게 되어있다

Array 클래스는 배열을 생성하는 기능, 추가, 삭제, 찾기 등의 기능을 담고 있다

주요 프로퍼티

프로퍼티	설명
length	배열의 크기를 알 수 있다

주요 메소드

메소드	설명
concat()	배열에 다른 배열이나 값을 연결하여 새로운 배열을 반환
indexOf()	배열 요소의 인덱스 값을 반환, 해당 요소가 없을 경우 -1 반환
join()	지정된 구분 문자열로 배열 요소들을 이어 붙여 새로운 배열 반환
pop()	마지막 배열 요소를 반환
push()	새로운 배열 요소를 마지막 위치에 추가
reverse()	배열 요소의 순서를 반대로 바꿈
slice()	배열의 일부분을 반환
sort()	배열을 내림차순 또는 오름차순으로 정렬
splice()	배열 요소를 추가, 삭제, 교체

배열 생성하기

리터럴 방식으로 배열을 생성하면 Array 객체를 생성하여 반환받아 사용하게 된다. 리터럴 방식이 더 편하기 때문에 리터럴 방식을 많이 사용한다.

리터럴 방식

```
<script>
var arr = [ "Apple", "Banana", "Cherry" ];
</script>
```

Array 객체 생성 방식

```
<script>
    var arr = new Array("Apple", "Banana", "Cherry");
</script>
```

배열 길이 알아내기 – length 프로퍼티

length 프로퍼티를 이용해 배열의 요소 개수를 알 수 있다

```
<script>
    var coms = ["com1", "com2", "com3"];
    alert(coms.length); // 3
</script>
```

특정 위치 배열 요소 접근

```
배열변수[index]
```

배열의 N번째 요소에 접근하려면 배열 변수의 이름에 [ ]를 이용하여 인덱스를 지정하면 된다

배열의 요소를 전부 출력

```
<script>
var coms = ["com1", "com2", "com3"];
for(var i=0; i<coms.length; i++) {
        console.log(coms[i]);
}
</script>
```

배열을 문자열로 만들기 – join()

```
<script>
    var str = 배열.join([separator]);
</script>
```

separator : 배열 요소를 구분하기 위한 문자열. 구분자로 이용되며 필수 사항은 아니다

배열의 요소를 ,(콤마)를 구분자로 가지는 문자열로 변환

```
<script>
var coms = ["com1", "com2", "com3"];
alert(coms.join(","));  </script>
```

문자열을 배열로 만들기 – split()

```
<script>
var arr = 문자열.split(separator);
</script>
```

separator : 구분자

문자열에서 ,(콤마)를 구분자로 이용하여 배열로 옮기기

```
<script>
var coms = "com1,com2,com3";
var arr = coms.split(",");

for(var i=0; i<arr.length; i++) {
        console.log(arr[i]);
}
</script>
```

⑥ Object 클래스

객체는 객체리터럴을 사용하거나, new 연산자를 통해서 생성할 수 있다. 변수명에 자바스크립트 예약어를 사용할 수 없지만, 객체 속성명으로는 사용 가능하다. 물론 추천하지 않는다.

```
<script>
  var obj = {
        x : "x 속성값",
        "y" : "y(문자열) 속성값",
        z : function(){
            return "저는 속성에 지정된 함수입니다."
        }
    }
    console.log(obj);

    //1.2 new 연산자 통해 만들고, 속성추가.
    Var obj_ = new Object();
    obj_x = "x 속성값";
    obj_["y"] = "y(문자열) 속성값";
    obj_z = function(){
```

```
            return "저는 속성에 지정된 함수입니다."
        };
    console.log(obj_);

    //2. 속성값 접근하기
    console.log(obj.x);
    console.log(obj["x"]);
    //console.log(obj[x]);// Uncaught ReferenceError: x is not defined // 변수 x => undefined

    console.log(obj.y);
    console.log(obj["y"]);

    console.log(obj.z);
    console.log(obj["z"]());//함수 호출해서 문자열을 리턴받는다.
</script>
```

```
결과
    {x: "x 속성값", y: "y(문자열) 속성값", z: ƒ}
    {x: "x 속성값", y: "y(문자열) 속성값", z: ƒ}
    x 속성값
    x 속성값
    y(문자열) 속성값
    y(문자열) 속성값
    ƒ (){
            return "저는 속성에 지정된 함수입니다."
        }
    저는 속성에 지정된 함수입니다.
```

접근하려는 해당 속성이 존재하지 않을 경우, undefined를 리턴한다. 이때 || 연산자를 사용해서 해당 속성이 존재하지 않을 경우, 기본값을 지정할 수 있다.

```
<script>
    var v = obj.nonExist || "기본값";
    console.log(v); // 기본값
</script>
```

```
"기본값"
```

속성값을 갱신하거나, 추가, 삭제가 가능하다. 생성한 객체를 대상으로 다음 코드를 실행해 보자.

```
<script>
    obj.x = "속성값 갱신";
    console.log(obj.x);

    //존재하지 않는 속성 추가하기
    obj.add1 = "새로운 속성 추가";
    console.log(obj.add1);

    obj.add2 = {newAttr : "새로운 속성에 객체를 추가"}
    console.log(obj);
</script>
```

결과

속성값 갱신
새로운 속성 추가
{x: "속성값 갱신", y: "y(문자열) 속성값", add1: "새로운 속성 추가", add2: {newAttr: "새로운 속성에 객체를 추가"}, z: *f*}

위 예제에서 추가된 add2 속성처럼 추가될 속성의 값으로써 다른 객체를 전달하는 것 또한 가능하다.

Delete 연산자를 사용해서 해당 속성을 삭제도 가능하다.

```
<script>
    delete obj.add1;
    console.log(obj);
</script>
```

결과

{x: "속성값 갱신", y: "y(문자열) 속성값", add2: {newAttr: "새로운 속성에 객체를 추가"}, z: *f*}

객체는 참조방식으로 전달되므로, 복사되지 않는다.

```
<script>
 var dog1 = {
        name : "예삐",
        breed : "푸들",
```

```
            bark : function(){
                return "왈왈";
            }
        }

        var dog2 = dog1;//dog1을 dog2에 할당. 이때 참조값이 전달된다. 새로운 객체가 생성되지 않는다.
        Console.log(dog2);

        dog1.name = "뽀삐";
        console.log(dog2.name);//뽀삐

        //객체참조 테스트
        var a = {}, b = {}, c = {};
        //a, b, c는 각각 다른 빈 객체 참조.

        A = b = c = {};
        //a, b, c는 모두 같은 빈 객체 참조.
    </script>
```

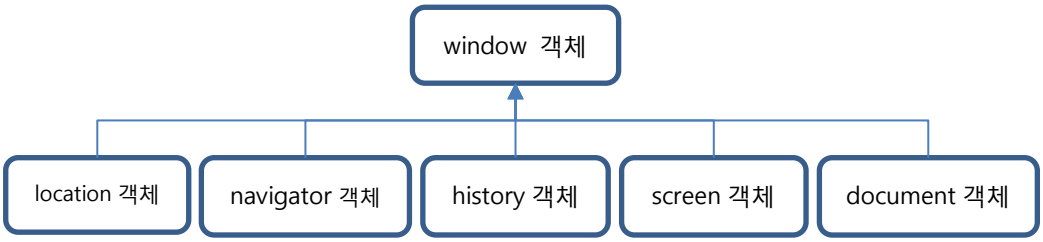
결과

```
{name: "예삐", breed: "푸들", bark: ƒ}
예삐
왈왈
뽀삐
```

13.2 자바스크립트 브라우저 객체 모델 (BOM)

BOM(Browser Object Model)이란 웹브라우저의 창이나 프래임을 추상화해서 프로그래밍적으로 제어할 수 있도록 제공하는 수단이다.

BOM은 전역객체인 Window의 프로퍼티와 메소드들을 통해서 제어할 수 있다.



① window 객체

브라우저 기반 자바스크립트의 최상위 객체이다.

var 키워드로 선언한 일반 변수도 모두 window 객체의 속성이 된다.

새로운 window 객체 생성

open() 메서드는 새로운 window 객체를 생성하는 메서드

dow.open([URL, name, features, replace]);

ript type="text/javascript">
dow.open();
dow.open('https//naver.com/', 'naver', 'width=500 height=500', true);
cript>

- 첫번째 매개변수 : HTML 페이지의 URL지정. 미지정시 about:blank로 열림.
- 두번째 매개변수 : 윈도우이름속성 혹은 새로운 윈도우의 타겟설정

속성값	윈도우 타겟
_blank	새창(기본값)
_parent	부모창
_self	현재창
_top	로딩가능한 아무 창

세번째 매개변수 : 윈도를 어떤 모양으로 출력할지 지정

속성명	설명	입력값
height	새 윈도우 높이	px 단위 값
width	새 윈도우 너비	px 단위 값
location	주소 입력창의 유무	yes, no, 1, 0
menubar	메뉴유무	yes, no, 1, 0
resizable	화면크기조절가능여부	yes, no, 1, 0
status	상태표시줄의 유무	yes, no, 1, 0
toolbar	브로우져 도구모음 유무	yes, no, 1, 0

네번째 매개변수 : 히스토리 목록에 현재 문서를 대체 혹은 새 항목 여부

```
  <body>
    <button id="blank">blnak</button>  
    <button id="parent">parent</button>  
    <button id="self">self</button>  
    <button id="top">top</button>
  <script>
    //버튼별 이벤트핸들러
    window.onload = function(){
      document.querySelector("#blank").onclick = function(){
          window.open("https://naver.com","_blank", "", true);
      }
      document.querySelector("#parent").onclick = function(){
        window.open("https://naver.com","_parent", "width=500, height=300");
      }
      document.querySelector("#self").onclick = function(){
        window.open("https://naver.com","_self","width=500, height=300", false);
      }
      document.querySelector("#top").onclick = function(){
        window.open("http://naver.com","_top", "width=500, height=300", true);
      }
    }
</scritp>
</body>
```

window 객체의 기본 메서드

자신의 형태와 위치를 변경할 수 있는 메서드를 제공

___By() 형태의 메서드는 현재 윈도를 기준으로 상대적으로 속성을 변화

___To() 형태의 메서드는 절대적인 기준으로 속성을 변화

메소드명	설명
moveBy(x,y)	윈도우 위치을 상대적으로 이동
moveTo(x,y)	윈도우 위치를 절대적으로 이동
resizeBy(x,y)	윈도우 크기를 상대적으로 지정
resizeBy(x,y)	윈도우 크기를 절대적으로 지정
scrollBy(x,y)	윈도우 스크롤 위치를 상대적으로 이동
scrollBy(x,y)	윈도우 스크롤 위치를 절대적으로 이동
focus()	윈도우에 초점 가지기
blur()	윈도우에 초점 제거
close()	윈도우 닫기

```
<script type="text/javascript">
var child = window.open('', '', 'width=300, height=200');
child.moveTo(0, 0);
setInterval(function(){
   child.moveBy(10, 10);
   }, 1000);
</script>
```

setInteval함수를 이용해서, child윈도우 객체를 1초에 우로10픽셀, 밑으로 10픽셀씩 이동시키는 예제이다.

② location 객체

프로토콜의 종류, 호스트 이름, 문서 위치 등의 정보

```
<script>
    output = "";
    for(var v in location){
        output += v + " : "+location[v]+"<br>"
    }
    document.body.innerHTML = output;    </script>
```

결과

replace : function () { [native code] }
assign : function () { [native code] }
href : http://localhost:9999/BOM/location.html
ancestorOrigins : [object DOMStringList]
origin : http://localhost:9999
protocol : http:
host : localhost:9999
hostname : localhost
port : 9999
pathname :/BOM/location.html
search :
hash :
reload : function reload() { [native code] }

location 객체의 기본속성

속성명	설명	예
href	문서의 url주소	http://localhost:9999/BOM/location.html
host	hostname+port	localhost:9999
hostname	호스트이름	localhost
port	포트번호	9999
pathname	디렉토리경로	/BOM/location.html
hash	앵커이름	#bookmark
search	요청매개변수	?param=1234
protocol	프로토콜종류	http:

location 객체의 기본메소드

메소드명	설명
assign(URL)	지정 URL로 페이지 이동
reload()	새로고침
replace(URL)	지정 URL로 페이지 이동(뒤로가기불가)

```
<body>
    <button id="btnReload">reload()</button>
    <button id="btnAssign">assign()</button>
    <button id="btnReplace">replace()</button>
<script>
    document.querySelector("#btnReload").onclick = function(){
        location.reload();
    }
    document.querySelector("#btnAssign").onclick = function(){
        location.assign("https://naver.com");
    }
    document.querySelector("#btnReplace").onclick = function(){
        location.replace("https://naver.com");
    }
</script>
```

해당 메소드들을 테스트해보자.

③ navigator 객체

브라우저의 정보를 제공하는 객체다. 주로 호환성 문제등을 위해서 사용된다.

속성명	설명	예
appName	웹브라우져 이름	파이어폭스, 크롬=>"Netscape"
appVersion	브라우져 버전	"5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.100 Safari/537.36"
userAgent	브라우져가 서버쪽으로 전송하는 USER-AGENT HTTP 헤더내용	"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.100 Safari/537.36"
platform	브라우져가 동작하는 운영체제에 대한 정보	"Win32"
cookieEnabled	브라우져가 cookie사용이 가능한지 여부	true/false
online	브라우져의 온라인 여부	true/false

```
console.dir(navigator);
```

```
Navigator
  appCodeName: "Mozilla"
  appName: "Netscape"
  appVersion: "5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.100 Safari/537.36"
  budget: BudgetService {}
  connection: NetworkInformation {downlink: 2.775, effectiveType: "4g", onchange: null, rtt: 75}
  cookieEnabled: true
  credentials: CredentialsContainer {}
  doNotTrack: null
  geolocation: Geolocation {}
  hardwareConcurrency: 4
  language: "ko"
  languages: (3) ["ko", "en-US", "en"]
  maxTouchPoints: 0
  mediaDevices: MediaDevices {ondevicechange: null}
  mimeTypes: MimeTypeArray {0: MimeType, 1: MimeType, 2: MimeType, 3: MimeType, 4: MimeType, length: 5}
  onLine: true
  permissions: Permissions {}
  platform: "Win32"
  plugins: PluginArray {0: Plugin, 1: Plugin, 2: Plugin, 3: Plugin, length: 4}
  presentation: Presentation {defaultRequest: null, receiver: null}
  product: "Gecko"
  productSub: "20030107"
  serviceWorker: ServiceWorkerContainer {controller: null, ready: Promise, oncontrollerchange: null, onmessage: null}
  storage: StorageManager {}
  usb: USB {onconnect: null, ondisconnect: null}
```

```
  userAgent: "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/61.0.3163.100 Safari/537.36"
  vendor: "Google Inc."
  vendorSub: ""
  webkitPersistentStorage: DeprecatedStorageQuota {}
  webkitTemporaryStorage: DeprecatedStorageQuota {}
  __proto__: Navigator
```

④ history 객체

사용자가 방문한 URL 히스토리를 가지고 있는 객체이다.

메소드명	설명	예
back()	히스토리리스트의 이전 url을 로드한다.	history.back();
forward()	히스토리리스트의 다음url을 로드한다.	history.forward()
go()	입력받은 수에 따라 앞뒤로 이동한다.	-1 : 이전 url 로딩(back()과 동일) 1 : 다음 url 로딩(forward()과 동일) -2 : 이전 두번째 url 로딩 0 :현재페이지 리로딩

아이디 num 인 input 태그에 숫자를 지정하고 history.go()를 실행해보자

```
    <body>
        <button id="back">back()</button><br>
        <button id="go">go()</button><input type="number" id="num"><br>
        <button id="forward">forward()</button><br>
        <button id="goNaver">히스토리쌓기::네이버</button>
    <script>
        document.querySelector("#back").addEventListener("click", function(){
            history.back();
        });
        document.querySelector("#go").addEventListener("click", function(){
            var num = document.querySelector("#num").value;
            history.go(num);
```

```
        });
        document.querySelector("#forward").addEventListener("click", function(){
            history.forward();
        });
        document.querySelector("#goNaver").addEventListener("click", function(){
            location.assign("https://naver.com");
        });
    </script>
    </body>
```

⑤ screen 객체

웹 브라우저의 화면이 아니라 운영체제 화면의 속성을 가지는 객체

속성명	설명
width	화면 너비
height	화면 높이
availWidth	실제화면에서 사용가능한 너비
availHeight	실제화면에서 사용가능한 높이
colorDepth	사용 가능한 색상수
pixelDepth	한 픽셀당 비트수

```
console.dir(screen);
```

```
Screen
    availHeight:1050
    availLeft:1920
    availTop:0
    availWidth:1920
    colorDepth:24
    height:1080
    orientation:ScreenOrientation {angle: 0, type: "landscapeprimary", onchange: null}
    pixelDepth:24
    width:1920
```

13.3 문서 객체 모델(DOM)

HTML 또는 XML 문서의 모든 요소에 접근하는 방법을 정의한 프로그래밍 인터페이스이다.
자바스크립트 코드에서 동적인 HTML을 만들기 위해 DOM 객체에 접근하여 문서구조, 스타일, 내용 등을 조작할 수 있다. DOM 은 구조화된 nodes 와 property 와 method 를 갖고 있는 objects로 된 문서를 표현한 것으로, 웹페이지를 스크립트 또는 프로그래밍 언어들이 사용할 수 있게 연결시켜주는 역할을 담당한다.
웹 페이지는 일종의 문서(document)다. 이 문서는 웹 브라우저를 통해 그 내용이 해석되어 웹 브라우저 화면에 나타나거나 HTML 소스 자체로 나타나기도 한다. 동일한 문서를 사용하여 이처럼 다른 형태로 나타날 수 있다는 점에 주목할 필요가 있다. DOM 은 동일한 문서를 표현하고, 저장하고, 조작하는 방법을 제공한다. DOM 은 웹 페이지의 객체 지향 표현이며, 자바스크립트와 같은 스크립팅 언어를 이용해 DOM 을 수정할 수 있다.
DOM은 프로그래밍 언어와 독립적으로 디자인되었다. 때문에 문서의 구조적인 표현은 단일 API를 통해 이용가능하다. 이 문서에서는 자바스크립트를 주로 사용하였지만, DOM 의 구현은 어떠한 언어에서도 가능하다.

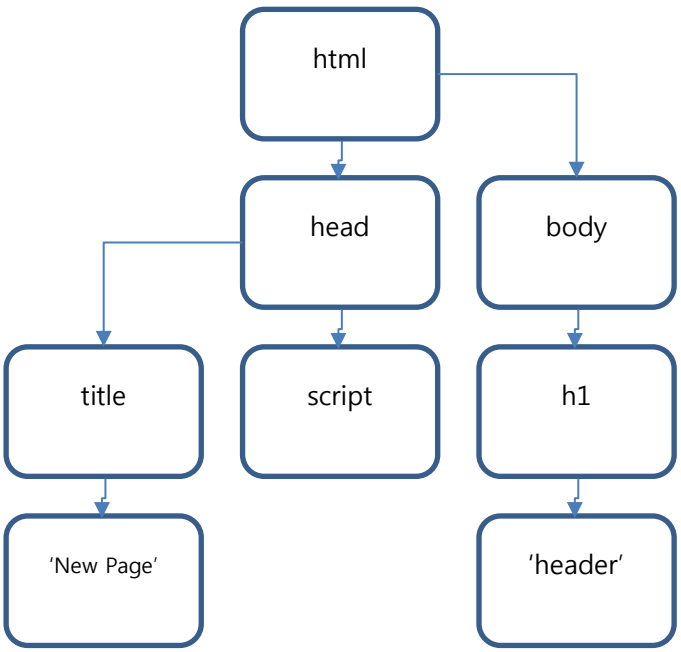
① DOM 구조

DOM은 정의 부분과 구현 부분으로 나뉘어져 있으며 정의 부분(명세)에는 실제 동작하는 구현 소스없이 웹 페이지 문서를 조작할 때 지켜야 하는 규약이 명시되어 있다. DOM 정의 부분을 만드는 곳은 웹 표준을 정의하는 W3C이다.
구현 부분은 각각의 브라우저에 존재하며 각 브라우저를 만드는 업체의 기술력을 바탕으로 DOM의 내부 동작 코드를 채워 구현한다. DOM 표준 정의에 맞지 않는 형식의 함수명이나 기능으로 구현을 해 놓은 브라우저일 경우 웹 표준을 지원하지 않는 브라우저라고 부른다.
DOM 을 사용하기 위해 특별히 해야할 일은 없다. 각각의 브라우저는 자신만의 방법으로 DOM 을 구현하였으며, 모든 웹 브라우저는 스크립트가 접근할 수 있는 웹 페이지를 만들기 위해 어느 정도의 DOM 을 항상 사용한다.
스크립트를 작성할 때 (인라인 <script> 요소를 사용하거나 웹 페이지 안에 있는 스크립트 로딩 명령을 사용하여), 문서 자체를 조작하거나 문서의 children 을 얻기 위해 document 또는 window elements 를 위한 API 를 즉시 사용할 수 있다.
DOM 프로그래밍은 아래처럼 window object 로 부터 alert() 함수를 사용하여 alert message 를 표시하는 매우 간단한 것일 수도 있고 새로운 content 를 작성하는 복잡한 DOM 이 될 수도 있다.

간단한 HTML 문서

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>New Page</title>
<script>
  window.onload = function() {
    var header = document.getElementById('header');
  };
</script>
</head>
<body>
<h1 id='header'>HEADER TEXT</h1>
</body>
</html>
```

DOM 구조



② DOM과 HTML 페이지

HTML 문서를 이용하여 웹 페이지를 작성하고 나서, 브라우저에서 읽어들이면 웹브라우저는 HTML 문서의 태그 정보를 파싱하여 DOM 객체를 만들게 된다. 만들어진 DOM 객체를 이용하여 웹페이지 화면을 구성하여 띄우는 것이다.

웹브라우저는 HTML 문서를 불러와 HTML 웹페이지 구성 요소인 노드들을 HTMLDocument 객체의 자식 객체들로 배치하여 트리 구조의 DOM 객체를 완성한다. 노드에는 <p>, 등의 태그들과 주석, 텍스트 등을 모두 해당된다.

③ 주요 DOM 객체

- **Node 객체**

노드를 다루는 기본 기능과 프로퍼티를 제공하며, 노드를 탐색하고 조작할 때 사용한다

노드 타입을 파악하거나 부모, 형제, 자식 노드를 알아내서 접근하거나, 추가, 삭제, 교체할 수 있다. DOM 객체의 최상위 객체이며 모든 하위 노드 객체들이 상속받는 객체이다.

- **Element 객체**

주석 노드와 텍스트 노드를 제외한 나머지 노드를 통합해서 칭하는 용어로, 태그 요소의 기본 기능과 프로퍼티를 제공하며 속성을 제거하거나 속성값을 구하고, 설정할 수 있는 기능을 제공한다. 이벤트와 관련된 기능을 사용할 수 있다.

- **HTMLElement 객체**

DOM 객체는 HTML 뿐만 아니라 XML을 위해서도 제공되는 객체로써 HTMLElement는 오직 HTML 문서에만 있는 노드를 통합해서 부르는 말이다. HTMLElement 객체는 태그의 ID, className 속성이 프로퍼티로 정의되어 있고, 오프셋 위치와 관련된 속성, 마우스 이벤트나 키보드 이벤트와 관련된 기능을 가지고 있다.

HTMLElement는 <a> 태그의 DOM 객체인 HTMLDivElement, <img> 태그의 DOM 객체인 HTMLImageElemnt, <body> 태그의 DOM 객체인 HTMLBodyElement와 같은 객체의 부모 객체이다.

- **Document 객체**

Node 객체의 하위 객체이며 HTML문서오 XML 문서의 Root 객체이다

엘리먼트 노드와 이벤트, 속성 노드, 텍스트 노드, 주석 등의 노드를 생성하는 기능과, id, className, tagName 등으로 특정 노드를 찾는 기능, 이벤트를 등록시키는 기능까지 제공하는 객체이다.

④ 문서 객체 만들기

DOM의 노드 생성 메소드

메소드	설명
createElement (tagName)	요소 노드를 생성
createTextNode (text)	텍스트 노드를 생성

h1 요소와 텍스트 노드 생성 스크립트

```
<script>
window.onload = function() {
  var header = document.createElement('h1');
  var textNode = document.createTextNode('Hello');
};
</script>
```

요소와 텍스트 노드를 생성하지만 아무런 일도 일어나지 않는다. 화면에 출력을 담당하는 부분은 <body> 태그인데 두 노드는 <body> 태그와 아무런 연관이 없기 때문이다. 화면에 두 노드를 출력되게 하려면 <body> 태그에 추가시켜 주어야 한다.

노드 연결 메소드

메소드	설명
appendChild(node)	객체에 노드를 연결

h1 요소와 텍스트 노드 생성 후 body에 연결하는 스크립트

```
<script>
window.onload = function() {
  // 노드 생성
  var header = document.createElement('h1');
  var t = document.createTextNode('Hello');

  // 노드 연결
  header.appendChild(t);
  document.body.appendChild(header);
};
</script>
```

요소와 텍스트 노드를 생성한 후 최종적으로 <body> 태그에 연결

완성된 HTML

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>New Page</title>

<script>
window.onload = function() {
  // 노드 생성
  var header = document.createElement('h1');
  var t = document.createTextNode('Hello');

  // 노드 연결
  header.appendChild(t);
  document.body.appendChild(header);
};
</script>

</head>
<body>
</body>
</html>
```

실행결과

- **속성 지정하기**

<img> 태그에 속성 지정하기

```
<script>
window.onload = function() {
  // 노드 생성
```

```
  var img = document.createElement('img');
   img.src =
 'https://www.google.com/images/branding/googlelogo/1x/googlelogo_color_272x92dp.png';
  img.width = 400;
  img.height = 100;

  // 노드 연결
  document.body.appendChild(img);
};
</script>
```

완성된 HTML

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>New Page</title>
<script>
    window.onload = function() {
      // 노드 생성
      var img = document.createElement('img');
      img.src =
     'https://www.google.com/images/branding/googlelogo/1x/googlelogo_color_272x92dp.png';
      img.width = 400;
      img.height = 100;

      // 노드 연결
      document.body.appendChild(img);
    };
</script>
</head>
<body>
</body>
</html>
```

실행결과

- **innerHTML 속성 사용하기**

  문서 객체의 innerHTML 속성은 문서 객체 내에 HTML을 설정하는 속성이다

  body 문서 객체의 innerHTML을 이용하여 노드를 추가할 수 있다

innerHTML을 이용해 목록 만들기

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>New Page</title>
<script>
      window.onload = function() {
     // 노드 생성
      var html = '';
      html += '<ul>';
      html += '<li>JavaScript</li>';
      html += '<li>DOM</li>';
      html += '<li>jQuery</li>';
    html += '</ul>';
   // 노드 연결
    document.body.innerH      TML = html;
};
</script>
</head>
<body>
</body>
</html>
```

⑤ 문서 객체 가져오기

이미 존재하는 HTML 태그를 자바스크립트로 가져오는 방법이다.

- **ID로 가져오기**

노드 추출 메소드

메소드	설명
getElementById (id)	ID 속성이 id인 문서 객체 추출

간단한 HTML

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
</head>
<body>
<h1 id="title_1">TITLE</h1>
<h1 id="title_2">JavaScript</h1>
</body>
</html>
```

ID값이 title_1과 title_2인 태그 추출 스크립트

```
<script>
window.onload = function() {
  var title_1 = getElementById('title_1');
  var title_2 = getElementById('title_2');
};
</script>
```

추출 후 속성값 변경

```
<script>
  window.onload = function() {
    var title_1 = document.getElementById('title_1');
    var title_2 = document.getElementById('title_2');
  title_1.innerHTML = 'JavaScript';
```

```
    title_2.innerHTML = 'with DOM';
    };
  </script>
```

실행결과

- **태그로 가져오기**

노드 추출 메소드

메소드	설명
getElementsByTagName (tagName)	tagName과 일치하는 문서 객체를 배열로 추출

간단한 HTML

```
  <!DOCTYPE html>
  <html>
  <head>
  <meta charset="UTF-8">
  <title>New Page</title>
  <script>
  window.onload = function() {
    var headers = document.getElementsByTagName('h1');
    headers[0].innerHTML = 'JavaScript';
    headers[1].innerHTML = 'with DOM';
  };
  </script></head>
  <body>
  <h1>TITLE</h1>
  <h1>JavaScript</h1>
  </body>
  </html>
```

⑥ 문서 객체 제거

노드 제거 메소드

메소드	설명
removeChild(child)	문서 객체의 자식 노드를 제거

간단한 HTML

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>New Page</title>
</head>
<body>
<h1>TITLE</h1>
<h1 id='remove'>DOM</h1>
<h1>JavaScript</h1>
</body>
</html>
```

노드 제거 스크립트

```
<script>
window.onload = function() {
  //노드 추출
  var remove = document.getElementById('remove');

  //노드 제거
  document.body.removeChild(remove);
};
</script>
```

실행한 화면에서는 DOM 문장이 출력이 안되는 것을 볼 수 있다

Chapter 4. jQuery

1. jQuery 개요

jQuery 는 2006년 존 레식(John Resig)에 의해서 소개되었으며, 클라이언트 측 솔루션 제작을 손쉽게 만들어주는 크로스 플랫폼 자바크스크립트 라이브러리로 출발했다.
jQuery는 복잡한 자바스크립트 코드를 간단하고 조작하기 쉽게 구현할 수 있다는게 특징이며, 오늘날 가장 인기있는 자바스크립트 라이브러리이다.
jQuery는 HTML 페이지의 조작에서부터 애니메이션, Ajax 통신, 이벤트 처리와 같은 JavaScript 개발에 관련된 거의 대부분의 기능을 지원하고 있다. 또한 차트 작성, 슬라이드 쇼, 엑셀과 같은 테이블도 **jQuery**와 플러그인을 이용함으로써 아주 간단한 코드로 구현 가능하다.
jQuery 플러그인 확장 라이브러리가 몇천에서 몇만 개씩 제공되고 있어 이것들을 이용함으로써 **jQuery** 기능을 제한없이 확장할 수 있다는 장점이 있다.

2. jQuery 기능

jQuery DOM

웹페이지의 HTML DOM(Document Object Model) 객체 조작을 위해서 노드 요소 및 노드 속성(아이디 및 클래스) 을 CSS 셀렉터(Selector) 를 기반으로 보다 쉽게 선택 및 탐색함으로써 HTML 문서 조작에 대한 편리한 기능들을 제공한다.

jQuery Ajax

Ajax 는 웹 페이지를 새로고침하지 않고도 서버와 데이터를 쉽게 주고 받을 수 있는 통신 기능을 제공하며 서버로부터 받은 데이터를 jQuery DOM과 연계할 수 있어 페이지의 일부를 업데이트할 수 있는 근본적인 방법으로, 동적인 페이지를 쉽게 제작할 수 있게 한다. jQuery 에서 지원하는 Ajax는 브라우저간 코드의 불일치를 해결할 수 있게 해 주었다.

jQuery 플러그인

jQuery의 기능을 확장하여 사용할 수 있도록 만들어진 자바스크립트 라이브러리들로, 다양한 기능들의 플러그인이 존재하며 필요한 기능이 있다면 미리 만들어져있는 오픈된 플러그인을 찾아서 활용할 수 있다

jQuery 특수효과 및 애니메이션

웹 페이지를 구성할 때 사용하는 효과(Effect) 및 애니메이션 기능을 기본적으로 제공하고 있으므로, DOM 요소와 연계하여 효과를 쉽게 적용할 수 있다

이벤트 처리

사용자가 요소를 클릭하거나, 키 입력을 하거나, 마우스를 클릭하면 브라우저는 발생한 이벤트에 해당하는 신호를 발생시키고, 이를 통해 사용자와 브라우저의 상호작용을 이용할 수 있다. 특히 사용자가 페이지에 대해 *어떤 작업*을 할 때마다 맞춤 이벤트를 이용해 응답할 수 있는데, 문제는 모든 브라우저가 동일한 방식으로 이벤트를 구현하지는 않는다는 것이다. 다행히 jQuery는 모든 이벤트에 대해 일관된 이름을 정의함으로써 이를 훨씬 쉽게 처리할 수 있다. 즉, 우리가 응답하려고 하는 이벤트에 대해 동일한 이름을 사용할 수 있고 이러한 이름은 모든 주요 브라우저에서 통한다.

3. jQuery 개발 환경 설정

jQuery 의 동작에 필요한 라이브러리는 jQuery 공식 사이트에서 CDN(Content Delivery Network : 콘텐츠 전송 네트워크)으로 제공되고 있다.
만약 오프라인 환경에서 동작되게 하고 싶다면 jQuery 공식 사이트에서 라이브러리 파일을 다운로드하면 된다.

CDN 이용

구글 Ajax API CDN : https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js
마이크로소프트 CDN : http://ajax.aspnetcdn.com/ajax/jQuery/jquery-3.2.1.js
jQuery CDN : https://code.jquery.com/jquery-3.2.1.min.js

jQuery CDN을 이용하여 웹페이지에 jQuery 라이브러리 삽입 예

```
<script type="text/javascript" src=" https://code.jquery.com/jquery-3.2.1.min.js"></script>
```

다운로드한 jQuery 라이브러리 파일 이용

jquery-3.2.1.min.js 파일을 다운받아 라이브러리 삽입하는 예제

```
<script type="text/javascript" src="jquery-3.2.1.min.js"></script>
```

jQuery 버전

jQuery 최신 버전을 사용할 경우, 버전별로 새로 추가된 기능과 더 이상 사용하지 않는

기능이 있다. 예를 들어 1.10.x 버전은 표준 및 비표준 모두 지원해서 IE 8 등의 브라우져를 지원하지만, 최신 버젼은 그렇지 않다. 최신 버젼을 사용하면서도, jQuery 구버젼의 기능들을 지원하기 위해서 jQuery migrate 를 추가하면 된다.
이전 버전으로 작성된 코드를 계속 사용하거나, 구버젼의 브라우져 지원을 위해 jQuery-migrate의 사용을 고려해 보자.

버전	대상
jQeury_Migrarte 1.4.1	1.9 버젼이하 코드를 1.9에서 3.0버젼에서 사용가능하게 해줌.
jQeury_Migrarte 3.0.0	3.0 버전이상으로 업데이트 지원. 1.x버전을 1.9버전이상으로 migration한 적이 있다면 사용.

또는 IE conditional comment를 사용해도 좋다. IE 9 이하 버전일 때는 1.12.4.js 를 로드하고, IE 9 버전보다 높거나, 다른 브라우저일 경우, 3.2.1.js를 로드해 사용하게 된다.

```
<!--[if lt IE 9]>
<script src="jquery-1.12.4.js"></script>
<![endif]-->
<!--[if gt IE 9]>
<script src="jquery-3.2.1.js"></script>
<![endif]-->
<!--[if !IE]>-->
<script src="jquery-3.2.1.js"></script>
<!--<![endif]-->
```

jQuery 의 모든 코드는 jQuery.ready() 메소드로 시작한다.

jQuery.ready() 메소드는 페이지 내의 코드를 모두 읽어들인 후에 실행하라는 의미이다.
자바스크립트의 window.onload 이벤트와 같다.

jQuery.ready() 메소드 사용방법

```
<script>
jQuery(document).ready(function() {
    // 코드
});
</script>
```

```
<script>
jQuery(function() {
```

```
  // 코드
});
</script>
```

```
<script>
$(document).ready(function() {
  // 코드
});
</script>
```

```
<script>
$(function() {
  // 코드
});
</script>
```

jQuery 와 $ 는 jQuery 코어라고 부르며 jQuery를 사용하겠다는 의미를 가지고 있다. 간편하게 사용할 수 있는 $ 를 주로 사용하지만 jQuery 이외의 자바스크립트 라이브러리를 사용할 경우 충돌이 일어날 수 있으므로 jQuery를 사용해야 한다. jQuery 코어를 간편하게 변수에 담아 사용할 수도 있다.

jQuery 코어를 J 변수에 담아 코어($)대신 사용하는 예제

```
<script type="text/javascript">
var J = jQuery;
J(document).ready(function() {
alert("안녕하세요");
});
</script>
```

jQuery를 사용하는 HTML문서 기본 구조

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title> </title>
```

```
<script type="text/javascript" src="http://code.jquery.com/jquery-3.2.1.min.js"></script>
<script type="text/javascript">
    $(document).ready(function() {
      alert("안녕하세요");
    });
</script>
</head>
<body>
</body>
</html>
```

4. 노드 찾기

노드 추가 삭제, 이동 등의 작업을 하려면 우선 노드를 찾아야 한다
스타일이나 속성을 다루거나 이벤트를 등록하고 제거할 때도 마찬가지로 노드 찾기가 선행되어야 한다

```
<body>
<div id="frontEnd">
        <h2>Front End</h2>
        <ul>
            <li class="myFavourite">html</li>
            <li class="myFavourite">css</li>
            <li class="js">JavaScript
                <ul>
                    <li    id="JS_CORE"><a    href="./JavaScript_Core.html">JavaScript
Core</a></li>
                    <li id="DOM"><a href="./DOM.js">DOM</a></li>
                    <li id="BOM"><a href="./BOM.js">BOM</a></li>
                </ul>
            </li>
            <li class="js">jQuery</li>
        </ul>
    </div>
    <div id="backEnd">
```

```
        <h2>Back End</h2>
        <ul>
            <li class="java myFavourite">java</li>
            <li class="java">jsp/servlet</li>
            <li class="myFavourite">oracle</li>
        </ul>
    </div>
</body>
```

일반 노드 찾기

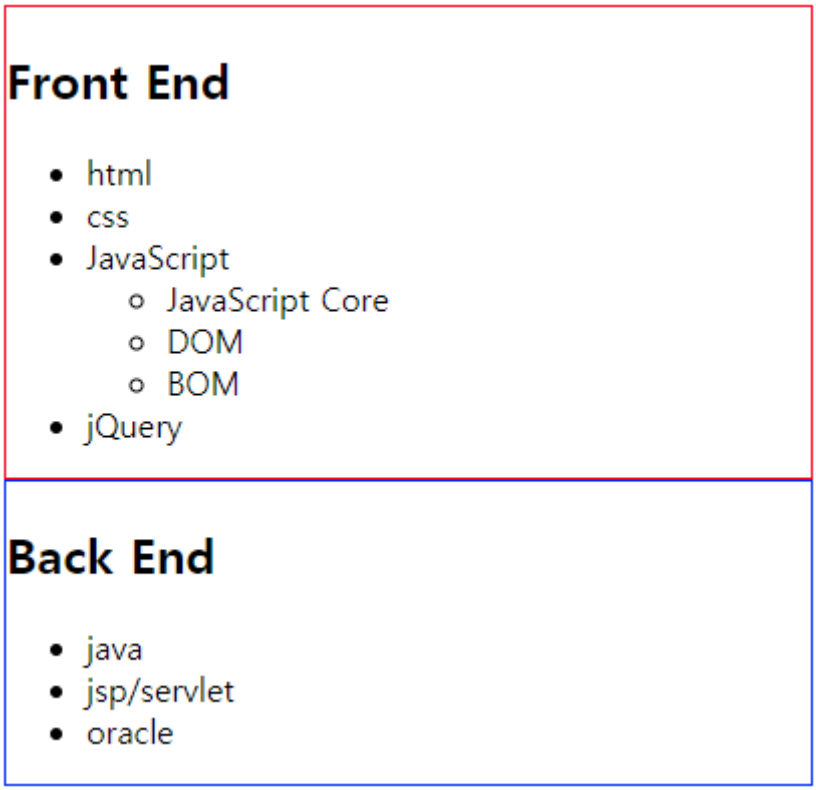
아이디 이름으로 노드 찾기

```
$("#아이디 이름")
```

아이디 이름 앞에 '#' 특수문자를 붙여 매개변수로 사용하여 $( ) 함수를 호출하면 해당 아이디의 노드를 찾는다

아이디 셀렉터 예제

```
<script>
$(document).ready(function() {
   $("#frontEnd").css("border","1px solid #ff00ff");
   $("#backEnd").css("border","1px solid #00ffff");
});
</script>
```



클래스 이름으로 노드 찾기

```
$(".아이디 이름")
```

클래스 이름 앞에 '.' 를 붙여 $( )함수의 매개변수로 사용해 호출하면 해당 클래스의 노드를 찾는다

클래스 셀렉터 예제

```
<script>
$(document).ready(function() {
$(".java").css("background", "wheat");
});
</script>
```

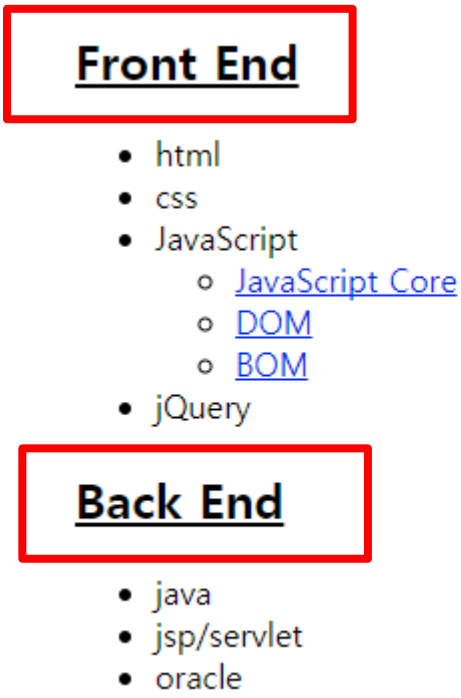
태그 이름으로 노드 찾기

```
$(”태그 이름”)
```

태그 이름을 매개변수로 $( )함수를 호출

태그 셀렉터 예제

```
<script>
$(document).ready(function() {
   $("h2").css("text-decoration","underline");
});
</script>
```



속성 이름으로 노드 찾기

$("속성옵션")

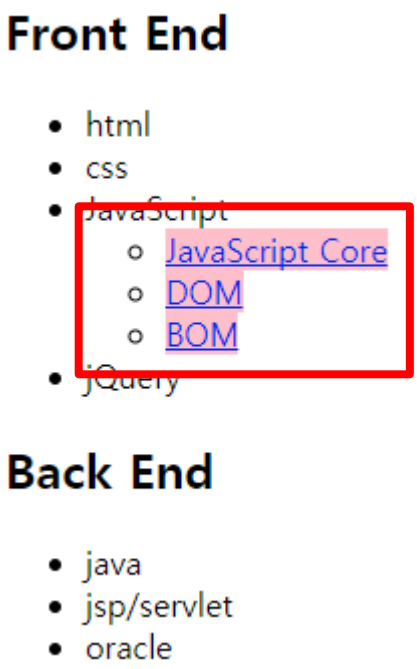
속성 옵션을 매개변수로 이용하여 $( ) 함수를 호출하여 해당 속성의 노드를 모두 찾을 수 있다. 속성 옵션에는 다음과 같이 있다.

$("E[A]")	속성 A를 포함한 모든 E 노드
$("E[A=V]")	속성 A의 값이 V인 모든 E 노드
$("E[A^=V")	속성 A의 값이 V로 시작하는 모든 E 노드
$("EA$=V")	속성 A의 값이 V로 끝나는 모든 E 노드
$("E[A*=V")	속성 A의 값이 V를 포함하는 모든 E 노드

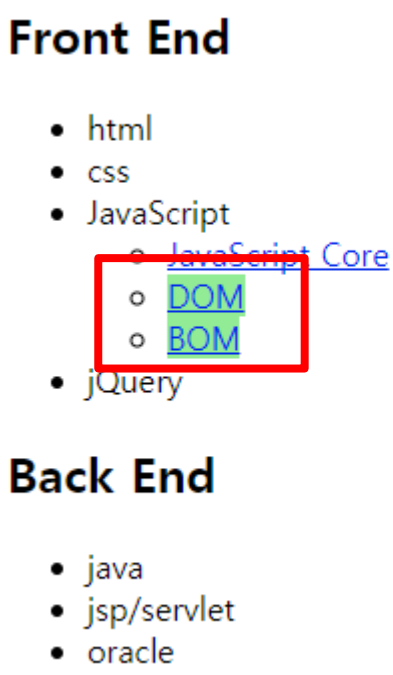
href 속성을 가진 노드나, href의 속성중 특정값을 필터링해서 노드를 검색해보자.

<script>

```
$(document).ready(function() {
   $("[href]").css("background", "pink");
});
</script>
```




```
<script>
$(document).ready(function() {
   $("[href]").css("background", "pink");
   $("[href$=js]").css("background", "lightgreen");
});
</script>
```



찾은 노드 다루기

$( ) 함수를 통해 노드를 찾으면 찾은 노드에 해당하는 객체를 반환한다
찾은 노드를 변수에 담아 활용하거나 반환된 객체를 이용해 노드를 다룰 수 있다

myFavourite 클래스 노드를 $myF 변수에 담기

```
<script>
var $myF = $(".myFavourite");
</script>
```

찾은 노드 개수 구하기 – length 프로퍼티

jQuery 의 length 프로퍼티를 이용하여 jQuery객체 내부의 노드 개수를 구한다

div 태그의 개수 구하기

```
<script>
$(document).ready(function() {
```

```
  alert("$myF.length = "+$myF.length);});
</script>
```

n번째 노드 접근하기 – eq() 메소드

jQuery의 eq() 메소드를 이용하여 n 번째 노드에 접근한다

첫 번째 노드가 0부터 인덱스를 가진다

ul의 내부 노드(li 태그) 중 2번째 노드의 border CSS 스타일 지정

```
<script>
$(document).ready(function() {
var $myF_2 = $myF.eq(2);
    $myF_2.css("border", "1px solid gray");
});
</script>
```

jQeury는 각메서드가 해당객체를 리턴하기 때문에 연속적으로 메소들를 연결할 수 있다. 이를 메소드체이닝 Method Chaining이라고 한다.

```
<script>
$(document).ready(function() {
// 한 줄로 표현 가능
 $(".myFavourite").eq(2).css("border", "1px solid gray");
});
</script>
```

DOM 객체 접근하기 – get() 메소드

get() 메소드를 이용하면 jQuery 객체 내부의 n번째 노드의 DOM 객체를 얻을 수 있다

ul의 내부 노드(li 태그) 중 1번째 노드의 DOM 객체를 알아내 스타일 변경

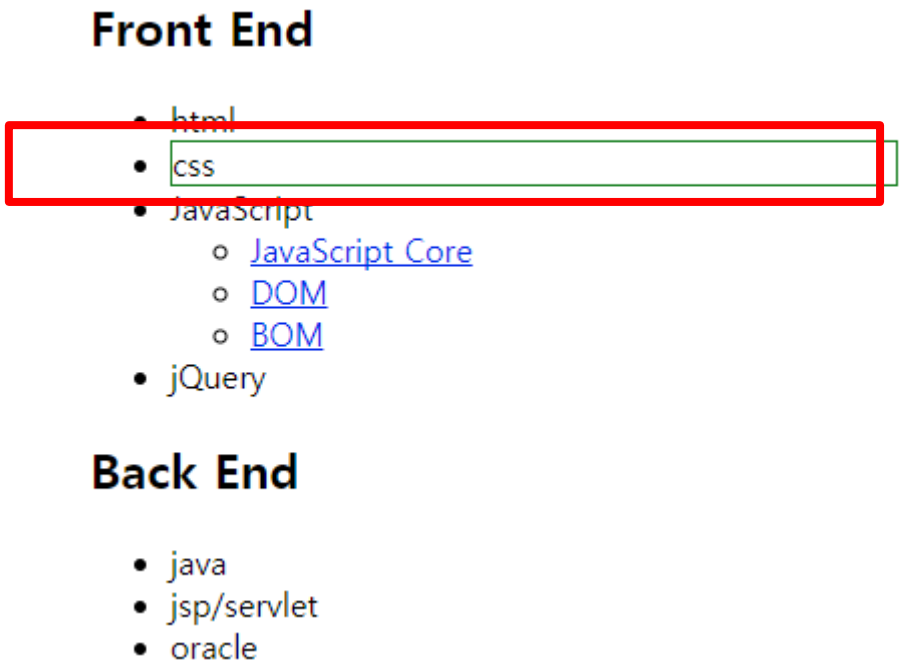
```
<script>
$(document).ready(function() {
   var $list = $("ul li");
   var list_1 = $list.get(1); // DOM 객체는 jQuery 객체가 아니기 때문에 변수 앞에 $를 붙이지 않음

   list_1.style.border = "1px solid green";
});
</script>
```

```
<script>
$(document).ready(function() {
// jQuery 객체를 배열처럼 접근하여 DOM 객체를 알아올 수 있다(get() 메소드와 동일)
var $list = $("ul li");
```

```
var list_1 = $list[1];

list_1.style.border = "1px solid green";
});
</script>
```



순차적으로 접근하기 – each() 메소드

each() 메소드를 사용하여 찾은 노드를 순차적으로 접근할 수 있다. 이때 인자로 받는 함수는 3가지 타입이 있다.

function(){} : 인자 없음.

function(index){} : 각 노드의 index값을 매개변수로 사용.

function(index, element){} : 각 노드의 index값과 DOM객체를 매개변수로 사용.

또, 각 노드를 조회할 때, 해당노드는 this(DOM객체), $(this)(jQuery객체)를 이용해서 접근할 수 있다.

```
<script>
```

```
$(document).ready(function() {
    var $list = $("ul li");
$list.each(function(){
        console.log($(this).text());
    });

    $list.each(function(index){
        console.log(index);
    });

  $list.each(function(index, element) {
    document.write(element.innerText);
//document.write(element.textContent);

    //document.write($(element).text());
  });
});
</script>
```

```
html
css
JavaScript
    JavaScript Core
    DOM
    BOM
JavaScript Core
DOM
BOM
jQuery
java
jsp/servlet
oracle
0
1
2
```

```
3
4
5
6
7
8
9
```

html
css
JavaScript JavaScript Core DOM BOM
JavaScript Core
DOM
BOM
jQuery
java
jsp/servlet
oracle

document.write() 스크립트는 모두 주석처리후에 다음 코드를 실습해보자.

자손 노드 중 특정 노드 찾기 – find() 메소드

찾은 노드의 자손 노드 중에서 특정 노드를 찾고 싶을 때 사용

```
$대상노드.find("선택자")
```

현재 노드는 제외되며 해당 노드의 자손들 중에서 조건에 맞는 객체를 찾는다

id가 content인 노드의 자손들 중 className이 redBox인 객체들의 border 설정

```
<script>
$(document).ready(function() {
  $("#backEnd").find(".java").eq(0).css("background", "yellow");
});
</script>
```

자식 노드 찾기

특정 노드의 바로 아래에 위치하고 있는 노드를 자식 노드라고 하며 하위 노드의 하위 노드는 자식노드라고 하지 않는다

모든 자식 노드 찾기 – children()

```
$대상노드.children()
```

특정 노드의 바로 하위에 위치한 모든 자식 노드를 찾고 싶을 때 사용

id가 test인 노드의 모든 자식노드의 border 설정

```
<script>
$(document).ready(function() {
   $("#test").children().css("border", "3px solid #0f0");
});
</script>
```

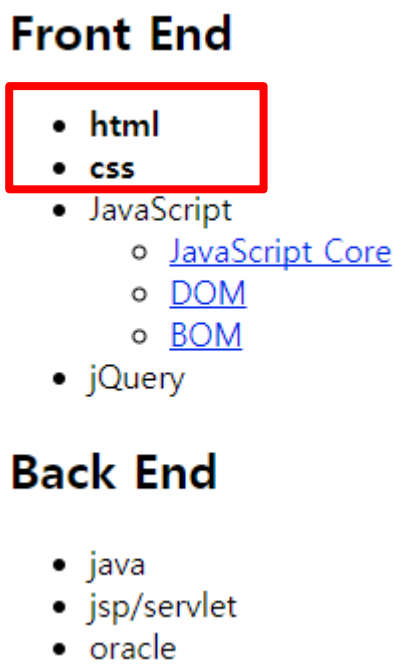
특정 자식 노드 찾기 – children()

```
$대상노드.children("선택자")
```

선택자를 이용하여 특정 자식 노드만을 찾고 싶을 때 사용

id가 frontEnd인 노드의 자식중, ul노드를 찾고, 그 자식노드 중 className이 myFavourite인 노드들의 폰트를 bold처리하기.

```
<script>
$(document).ready(function() {
$("#frontEnd").find("ul").children(".myFavourite").css("font-weight","bold");});
</script>
```



부모 노드 찾기

특정 노드를 감싸고 있는 바로 위의 상위 노드를 부모 노드라고 한다

조상 노드는 특정 노드의 모든 상위 노드를 말한다

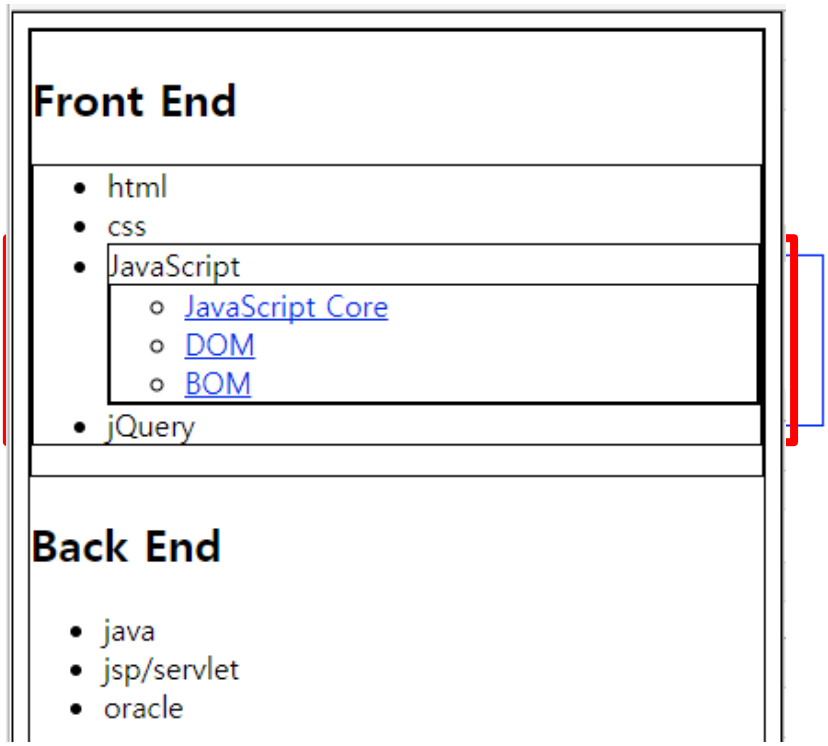
부모 노드 찾기 – parent()

```
$대상노드.parent()
```

특정 노드의 바로 상위에 존재하는 부모 노드를 찾을 때 사용

id가 DOM인 노드의 부모 노드ul, 다시 그 부모노으 li노드의 border 설정

```
<script>
$(document).ready(function() {
$('#DOM').parent().parent().css("border","1px solid blue");
});
</script>
```



조상 노드 – parents()

```
$대상노드.parents()
```

특정 노드의 모든 상위 노드, 즉 조상 노드들을 찾을 때 사용

id가 DOM인 노드의 조상 노드들 border 설정

```
<script>
$(document).ready(function() {
$("#DOM").parents().css("border","1px solid black");
});
</script>
```

id가 DOM인 노드의 모든 부모노드이기 때문에 ul, li, ul, div, body노드 모두 border처리되었다.

형제 노드 찾기

같은 깊이에 존재하는 노드를 형제 노드라고 한다. 즉 같은 부모 노드를 공유한다.

이전 형제 노드 찾기 – prev()

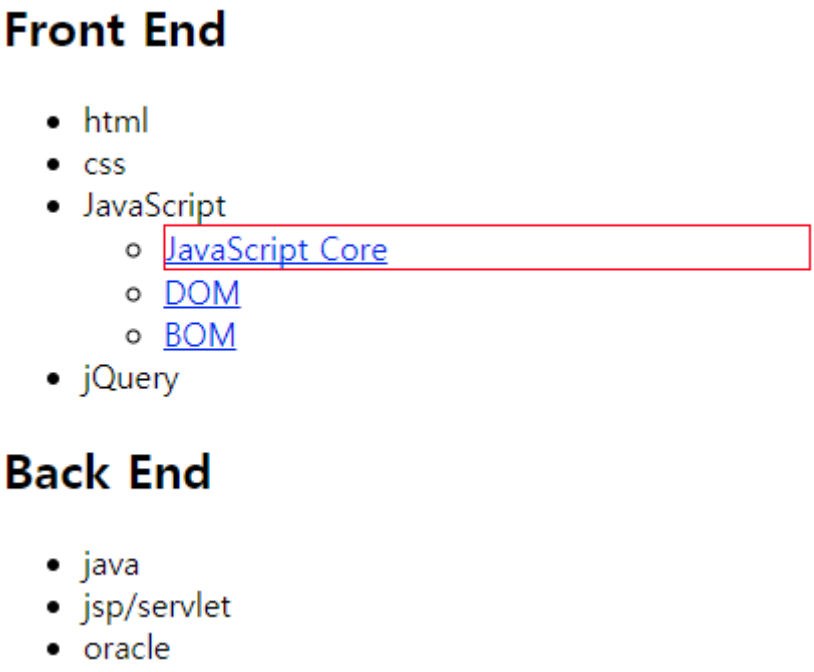
```
$대상노드.prev()
```

특정 노드의 이전 형제 노드를 찾을 때 사용

두 노드 앞의 형제 노드를 찾고 싶다면 prev()를 두 번 사용한다

id가 DOM인 노드의 이전 형제 노드 border 설정

```
<script>
$(document).ready(function() {
   $("#DOM").prev().css("border", "2px solid #f00");
});
</script>
```



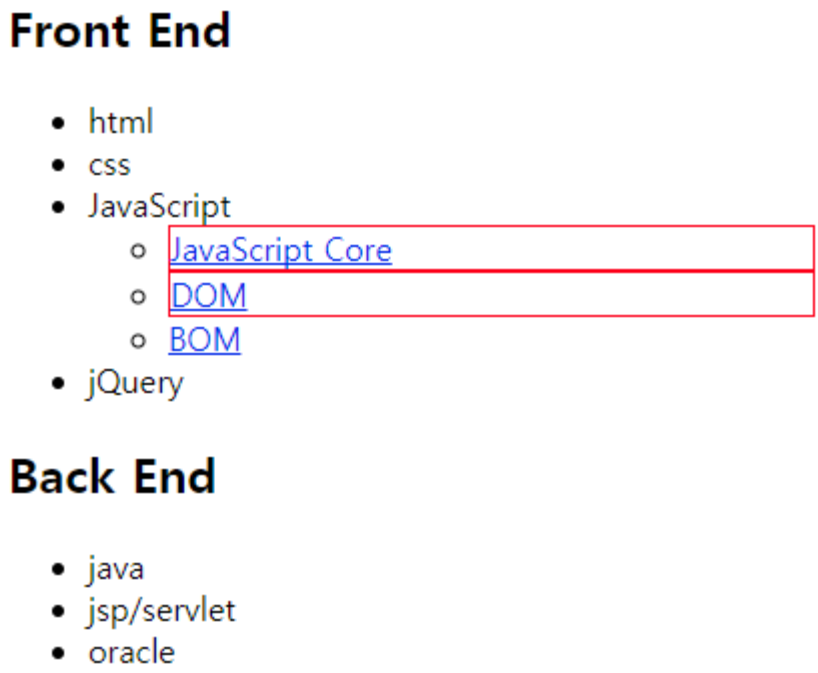
모든 이전 형제 노드 찾이 – prevAll()

```
$대상노드.prevAll()
```

특정 노드의 모든 이전 형제 노드를 찾을 때 사용

id가 BOM인 노드의 모든 이전 형제 노드들 border 설정

```
<script>
$(document).ready(function() {
   $("#BOM").prevAll().css("border", "1px solid #00f");
});
</script>
```



다음 형제 노드 찾기 – next()

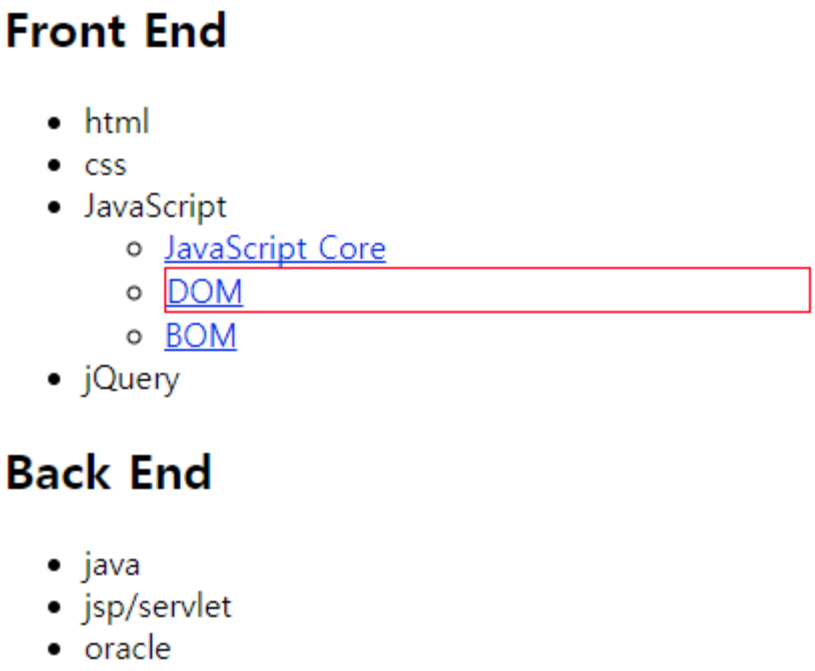
```
$대상노드.next()
```

특정 노드의 다음 형제 노드를 찾을 때 사용

id가 test인 노드의 다음 형제 노드 border 설정

```
<script>
$(document).ready(function() {
$("#JS_CORE").next().css("border","1px solid #f00");
});
```

```
</script>
```



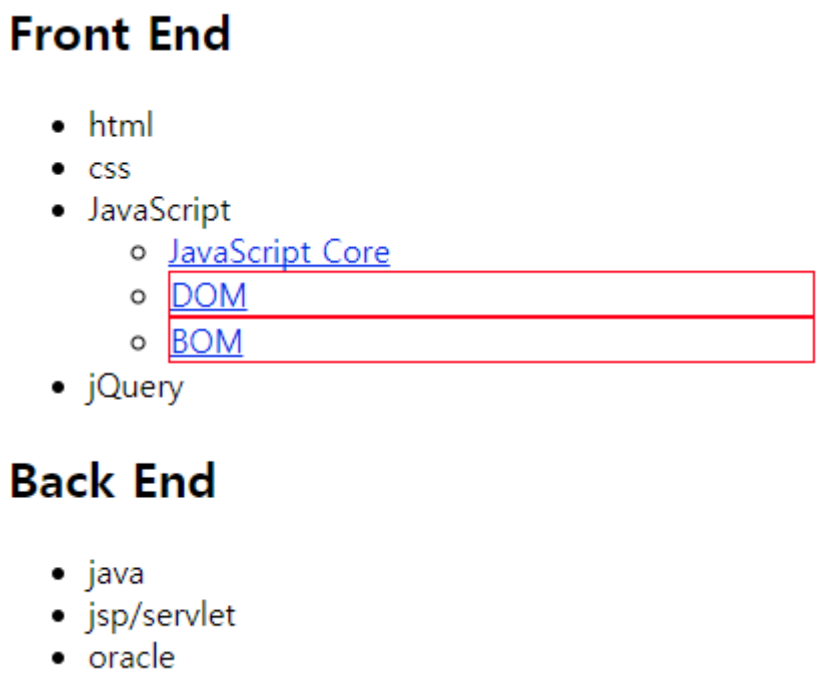
모든 다음 형제 노드 찾이 – nextAll()

```
$대상노드.nextAll()
```

특정 노드의 모든 다음 형제 노드를 찾을 때 사용

id가 JS_CORE인 노드의 모든 다음 형제 노드들 border 설정

```
<script>
$(document).ready(function() {
   $("#test").nextAll().css("border", "1px solid #00f");
});
</script>
```



5. 노드 생성/추가/삭제/이동

노드 생성/추가

노드 생성

```
var $신규노드 = $("신규DOM");
```

$( ) 함수 내부에서 매개변수로 받은 태그 노드 정보를 파싱해 태그에 맞는 HTMLElement 객체를 생성한다. 자바스크립트 DOM 객체로 만들어진 정보는 jQuery DOM 객체로 변환하여 반환한다.

div 노드 생성

```
<script>
$(document).ready(function() {
        var $div = $("<div>안녕, 나 제이쿼리야~</div>");
        $("body").append($div);
});
</script>
```

div 노드 생성을 자바스크립트 DOM으로 구현

```
<script>
//자바스크립트로 구현하기
		var div = document.createElement("div");
		var txt = document.createTextNode("안녕, 나 자바스크립트야~");
		div.appendChild(txt);
		document.body.appendChild(div);
</script>
```

안녕, 나 제이쿼리야~
안녕, 나 자바스크립트야~

jQuery 코드를 이용하여 생성하는 방법은 매우 간단하지만 자바스크립트 DOM을 이용하여 만들면 상당히 복잡해진다. 하지만 jQuery 코드의 내부에서는 자바스크립트 DOM을 이용하여 구현하고 있다는 사실을 알고 있어야 한다.

다음 코드를 새로생성하고 다음 예제를 실습해보자.

```
<body>
    <div id="content">
        <ul class="menu">
            <li>menu1</li>
            <li>menu2</li>
            <li id="select">menu3</li>
            <li>menu4</li>
            <li>menu5</li>
            <li>menu6</li>
        </ul>
    </div>
    <button id="btnAdd">추가</button>
</body>
```

생성한 노드를 첫 번째 자식 노드로 추가

```
$부모노드.prepend($추가노드)
```

```
$추가노드.prependTo($부모노드)
```

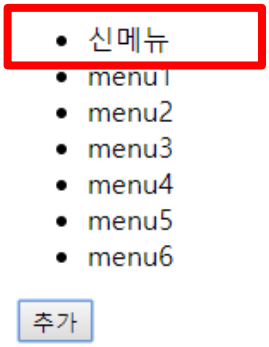
prepend()와 prependTo()는 같은 기능을 하며 부모노드와 추가할 노드의 순서만 다르다

버튼의 click이벤트핸들러는 다음과 같이 추가한다.

```
$대상노드.이벤트명(이벤트핸들러함수);
```

아이디가 btnAdd를 클릭할 때, 발생하는 이벤트핸들러를 다음과 같이 생성하고, 그안에 새 노드를 추가하는 코드를 추가하였다. 클릭이벤트가 발생할 때 마다 다음과 같이 기존 메뉴 상위에 새 li노드가 추가되고 있다.

```
<script>
$(document).ready(function() {
var $newMenu = "<li>신메뉴</li>";
                $("#btnAdd").click(function(){
                    $(".menu").prepend($newMenu);
//$($newMenu).prependTo($(".menu"));
                });
});
</script>
```



prependTo를 사용할때는 문자열로 생성한 $newMenu를 다시 제이쿼리객체화 해야 하기 때문에, $($newMenu)처럼 생성하고, prependTo를 구현한다.

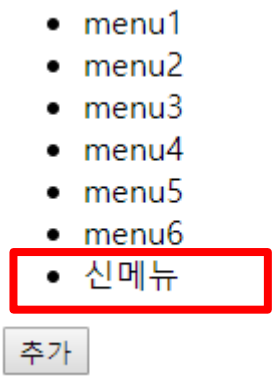
생성한 노드를 마지막 자식 노드로 추가

```
$부모노드.append($추가노드)
```

```
$추가노드.appendTo($부모노드)
```

append()와 appendTo()는 같은 기능을 하며 부모노드와 추가할 노드의 순서만 다르다

```
<script>
$(document).ready(function() {
var $newMenu = "<li>신메뉴</li>";
                $("#btnAdd").click(function(){
$(".menu").append($newMenu);
                    //$($newMenu).appendTo($(".menu"));
});
});
</script>
```

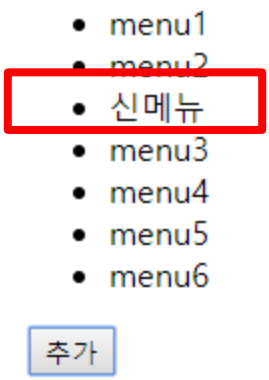


생성한 노드를 특정 노드의 이전 위치에 노드 추가

```
$추가노드.insertBefore($기준노드)
$기준노드.before($추가노드)
```

신규 노드를 특정 노드의 이전 형제 노드로 추가할 경우 사용

```
<script>
$(document).ready(function() {
var $newMenu = "<li>신메뉴</li>";
                $("#btnAdd").click(function(){
$($newMenu).insertBefore($("#select"));
        //$("#select").before($newMenu);});
});
</script>
```



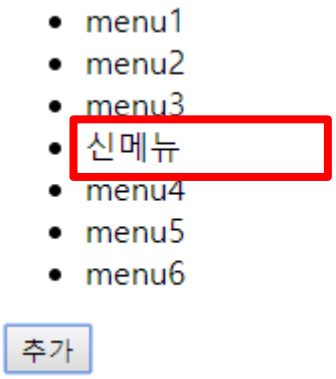
생성한 노드를 특정 노드의 다음 위치에 노드 추가

```
$추가노드.insertAfter($기준노드)
$기준노드.after($추가노드)
```

신규 노드를 특정 노드의 다음 형제 노드로 추가할 경우 사용

```
<script>
$(document).ready(function() {
var $newMenu = "<li>신메뉴</li>";
```

```
                $("#btnAdd").click(function(){
$("#select").after($newMenu);
                        //$($newMenu).insertAfter($("#select"));
});
});
</script>
```



노드 이동

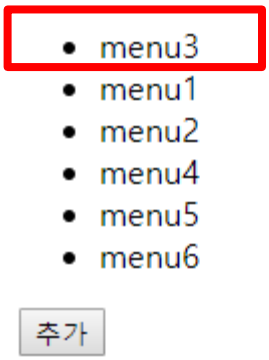
노드를 추가할 때 사용 하는 함수와 같은 형태의 함수를 사용하여 노드를 이동시킬 수 있다

추가노드 대신에 이동시킬 노드를 사용하면 추가하는 대신 노드를 이동시키게 된다

노드를 첫 번째 자식 노드로 이동

```
$부모노드.prepend($이동노드)
$이동노드.prependTo($부모노드)
```

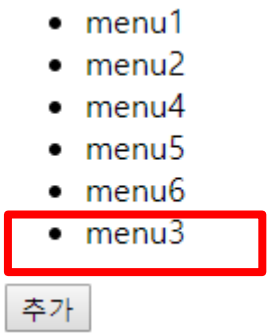
```
<script>
$(document).ready(function() {
                $("#btnAdd").click(function(){
var $move = $("#select");
$(".menu").prepend($move);
//$move.prependTo($(".menu"));
});
});
</script>
```



노드를 마지막 자식 노드로 이동

```
$부모노드.append($이동노드)
$이동노드.appendTo($부모노드)
```

```
<script>
$(document).ready(function() {
                $("#btnAdd").click(function(){
var $move = $("#select");
$(".menu").append($move);
//$($move).appendTo($(".menu"));});
});
</script>
```

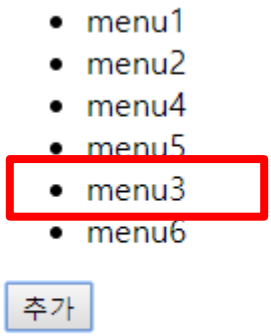


생성한 노드를 특정 노드의 이전 위치로 이동

```
$이동노드.insertBefore($기준노드)
$기준노드.before($이동노드)
```

특정 노드의 이전 형제 노드로 이동시킬 경우 사용

```
<script>
$(document).ready(function() {
                $("#btnAdd").click(function(){
var $move = $("#select");
$(".menu").prepend($move);
//$move.prependTo($(".menu"));
});
});
</script>
```

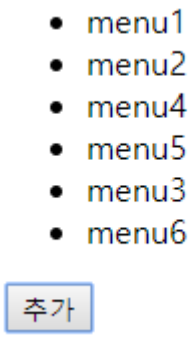


생성한 노드를 특정 노드의 다음 위치로 이동

```
$이동노드.insertAfter($기준노드)
$기준노드.after($이동노드)
```

신규 노드를 특정 노드의 다음 형제 노드로 시동시킬 경우 사용

```
<script>
$(document).ready(function() {
                $("#btnAdd").click(function(){
var $move = $("#select");
$("li").eq(4).after($move);
//$($move).insertAfter($("li").eq(4));});
});
</script>
```



노드 삭제

특정 노드 삭제

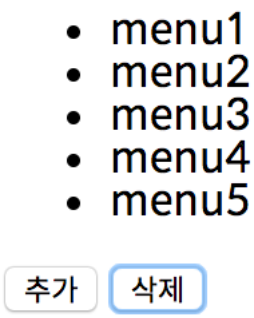
```
$대상.remove()
```

찾은 노드를 삭제한다

삭제버튼을 동적으로 추가하고, 클릭시 menu클라스의 자식노드li중 마지막 태그를 삭제해 보자.

```
<script>
$(document).ready(function() {
		//삭제버튼 동적 추가
		$("<button id='btnRemove'>삭제</button>").appendTo($("body"));

		//삭제버튼 클릭이벤트핸들러 추가
		$("#btnRemove").click(function(){
			var $menu_li = $(".menu").children();
			$menu_li.eq($menu_li.length-1).remove();
		});
});
</script>
```



삭제버튼을 클릭하면, 가장 마지막 li태그가 삭제된다.

모든 자식 노드 삭제

```
$대상.children().remove()
```

children() 메소드를 이용하여 모든 자식 노드를 찾아 remove() 메소드를 이용하여 모두 삭제할 수 있다

```
<script>
$(document).ready(function() {
		//삭제버튼 동적 추가
		$("<button id='btnRemove'>삭제</button>").appendTo($("body"));

		//삭제버튼 클릭이벤트핸들러 추가
		$("#btnRemove").click(function(){
			var $menu_li = $(".menu").children();
			$menu_li.remove();
		});
});
</script>
```

위와 같이 모든 li태그가 삭제되고, button만 남게된다.

노드 내용 읽기/변경

노드 내용을 문자열로 읽기

```
$대상.html()
$대상.text()
```

html() 메소드를 이용하여 노드의 내용을 마크업 태그까지 확인할 수 있다. 반환된 결과는 단순 문자열로 받게되며 자바스크립트 DOM 객체가 아니다.

text() 메소드를 이용하여 노드의 내용을 확인하면 마크업 태그를 제외한 내용만을 문자열로 반환받을 수 있다.

노드 내용 수정하기

```
$대상.html(수정할 태그 포함 문자열)
$대상.text(수정할 텍스트)
```

html() 메소드를 이용하여 노드의 내용으로 마크업을 포함하는 문자열로 수정할 수 있다. html() 메소드를 이용하여 마크업을 포함하는 문자열로 변경하면 마크업 문자는 태그문자로 인식하여 노드의 내용이 변경된다.
text() 메소드를 이용하여 노드의 내용을 수정하면 마크업 태그를 단순 문자열로 인식하여 변경된다.

실습을 위해 아래코드를 작성하자.

```
<body>
    <div id="container">hello, kh~</div>
    <button id='btnHTML'>html()</button>
    <button id='btnTEXT'>text()</button>
</body>
```

기존의 태그를 html(), text() 두가지 방법으로 가져와보자.

```
<script>
        jQuery(function(){
            console.log($('body').html());
            console.log($('body').text());
        });
</script>
```

결과

```
    <div id="container">hello, kh~</div>
    <button id="btnHTML">html()</button>
    <button id="btnTEXT">text()</button>
```

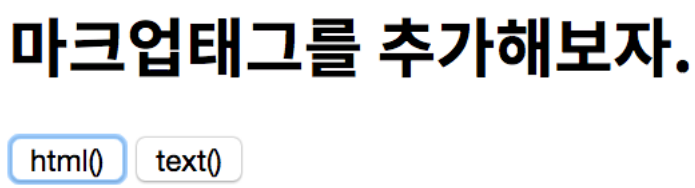
결과

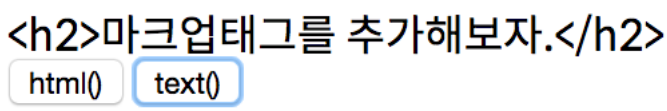
```
    hello, kh~
    html()
    text()
```

둘다 문자열로써 리턴하지만, html은 마크업정보를 함께 보여주지만, text는 태그안의 문자열만 리턴한다.

html(), text() 메소드를 활용해 텍스트를 추가해보자. markUpTag라는 텍스트는 태그정보가 포함된 문자열이다. html(), text()버튼에 각각 이벤트핸들러를 작성해서 두 메소드의 차이를 알아보자.

```
<script>
        jQuery(function(){
            var markUpTag = "<h2>마크업태그를 추가해보자.</h2>";
            $("#btnHTML").click(function(){
                $("#container").html(markUpTag);
            });
            $("#btnTEXT").click(function(){
                $("#container").text(markUpTag);
            });
        });
</script>
```





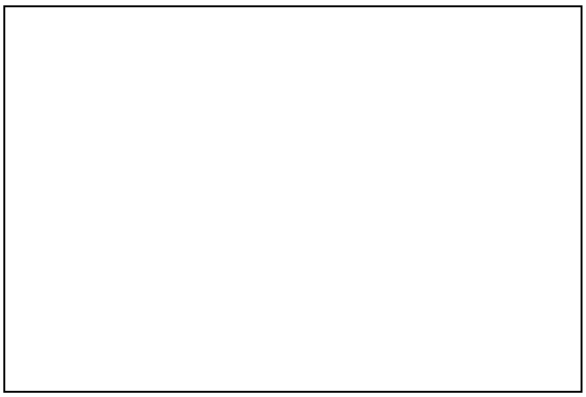
위와 같이 html()은 문자열의 마크업정보를 해석해 노드를 추가하지만, text()는 마크업정보를 문자열로 취급한다.

6. 스타일 다루기

jQuery를 이용하면 웹문서의 스타일을 동적으로 다룰 수 있다. 문서를 재실행하여 갱신해 줄 필요가 없다.

아래코드를 작성하자.

```
<head>
<style>
    div{
        width:300px;
        height:200px;
        border:1px solid black;
    }
    .border{
        border: 5px dashed blue;
    }
</style>
</head>
<body>
    <div id="test"></div>
</body>
```



스타일 값 구하기

```
$대상.css("스타일 속성이름")
```

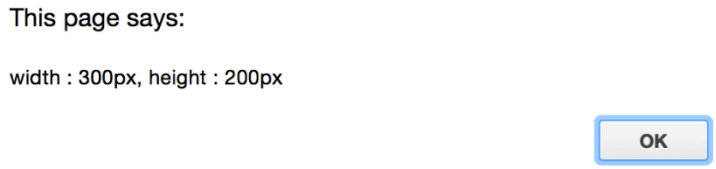
스타일 속성이름을 명시해 해당 스타일 속성값을 알아낸다

여러 스타일 속성을 한꺼번에 구하려면 [속성이름1, 속성이름2, ...] 과 같이 배열 형식을 이용한다

div 노드의 border 스타일 알아내기

```
<script>
$(document).ready(function() {
alert($("div").css("border"));
});
</script>
```

div 노드의 width, height 스타일 알아내기




```
<script>
$(document).ready(function() {
var divInfo = $("div").css(["width", "height"]);
alert("width : " + divInfo.width + ", height : " + divInfo.height);
});
</script>
```

스타일 설정하기

```
$대상.css("속성이름", "값")
$대상.css({
   속성이름: 값,
   속성이름: 값,
   ...
})
```

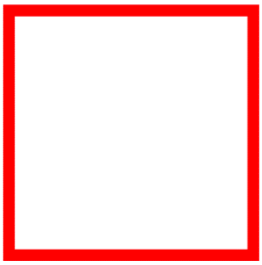
스타일 속성을 알아낼 따 사용한 css() 함수를 이용해 스타일을 지정할 수도 있다
첫 번째 매개변수로 속성이름을 두 번째 매개변수로 속성 값을 지정한다

#test 노드의 border 스타일 지정

```
<script>
$(document).ready(function() {
$("#test").css("border", "10px solid #f00");
});
</script>
```

#test 노드의 height와 width 스타일 한번에 지정

```
<script>
$(document).ready(function() {
$("#test").css({
width: 200,
height: 200
});
});
</script>
```



클래스 추가

다음과 같이 버튼엘레먼트를 추가하는 코드를 추가하자.

```
<script>
$(document).ready(function() {
		$("<button id='addClass'>클래스추가</button>").appendTo($("body"));
		$("<button id='removeClass'>클래스삭제</button>").appendTo($("body"));
});
</script>
```

```
$대상.addClass("클래스이름1 클래스이름2 ...")
```

addClass() 메소드를 이용하여 하나 이상의 클래스를 추가할 수 있다

#test 노드에 border 클래스 추가

클래스추가버튼을 추가하면, addClass메소드를 호출하는 이벤트 핸들러를 작성하자.

```
<script>
$(document).ready(function() {
		$("#addClass").click(function(){
			$("#test").addClass("border");
		});
}); </script>
```

클래스 삭제

```
$대상.removeClass() // 모든 클래스 삭제
$대상.removeClass("클래스이름1 클래스이름2 ...")
```

removeClass() 메소드를 이용하여 특정 클래스를 제거할 수 있다

클래스이름을 명시하지 않으면 모든 클래스를 제거하게 된다

```
<script>
$(document).ready(function() {
		$("#removeClass").click(function(){
			$("#test").removeClass("border");
		});
});
</script>
```

7. 속성

태그의 속성(Attribute)에는 일반 속성과 사용자 정의 속성이 있다.
id, class, <a>태그의 href, <img> 태그의 src 등이 일반 속성(Attribute)이다.
그 외에 data-value와 같이 사용자가 필요에 의해 만들어서 사용하는 속성을 사용자 정의 속성이라고 부른다.

아래 샘플 코드를 작성해 보자.
사용할 이미지는 동일한 경로상에 hr.png와 2ndimg.jpg 두 개가 있다.

```
<body>
    <a href="#hello" id="a">링크#hello</a> <br>
    <input type="checkbox" id="checked" checked="checked">      <br>
    <input type="checkbox" id="unchecked" > <br>
    <img id="sample_img" src="hr.png" width="200px"
            data-userattr1="사용자지정속성값" data-userattr2="123456789"> <br>
</body>
```

실행결과

링크#hello

일반 속성

attr(), prop() 두 메소드 모두 태그의 속성에 접근할 수 있다.
attr()은 html 태그에 기술되어 있는 속성을 반환하는 반면, prop()은 동적처리시 처리되는 값을 리턴한다.

```
var 변수 = $("대상").attr("속성이름")  또는 $("대상").attr("속성이름", "변경할값")
var 변수 = $("대상").prop("속성이름") 또는 = $("대상").prop("속성이름", "변경할값")
```

각 메소드의 사용법 및 차이를 알아보자.

```
var attr_img_src =   $("#sample_img").attr("src");
var prop_img_src =   $("#sample_img").prop("src");

console.log(attr_img_src);      //hr.png
console.log(prop_img_src);    //사용자 파일경로 /jQuery/hr.png

var attr_img_width = $("#sample_img").attr("width");
var prop_img_width = $("#sample_img").prop("width");

console.log(attr_img_width);    //200px
console.log(prop_img_width);   //200

var attr_href = $("#a").attr("href");
var prop_href = $("#a").prop("href");

console.log(attr_href);    //#hello
console.log(prop_href);   //사용자 파일경로 /jQuery/5_jquery_attr_prop.html#hello

var attr_checked = $("#checked").attr("checked");
var prop_checked = $("#checked").prop("checked");

console.log(attr_checked);    //checked
console.log(prop_checked);   //true

var attr_unchecked = $("#checked").attr("checked");
var prop_unchecked = $("#unchecked").prop("checked");

console.log(attr_unchecked);    //undefined
console.log(prop_unchecked);   //false
```

위와 같이 attr()은 html 태그에 적혀진 값(예: checked)을 리턴하는 반면, prop()은 처리되는 boolean값을 리턴한다.(예:false). 만약 checked 속성을 기술하지 않은 #unchecked 같은 경우 attr("checked")로 실행하면, undefined가 리턴된다.

속성을 동적으로 제어해 보자.

id가 btnModifyAttr인 버튼을 추가하고, 클릭 이벤트핸들러를 통해 동적으로 지정하는 코드를 작성하자.

```
$("<button id='btnModifyAttr'>속성변경</button>").appendTo($("body"));
$("#btnModifyAttr").click(function(){
                        $("#sample_img").attr("src", "2ndimg.jpg");
                        $("#checked").prop("checked", false);
                        $("#unchecked").prop("checked", true);
});
```

실행결과

링크#hello

속성변경

버튼 클릭후 아래와 같이 checked 속성이 반전되고, 2번째 이미지로 교체된다.

실행결과

링크#hello

속성변경

사용자정의 속성

사용자가 정의한 속성은 data()을 통해 접근하고, 변경할 수 있다.

```
$대상.data("data-를 제외한 속성이름")
```

인자로 전달할 속성이름은 data-을 제외하고 입력해야 하며, 사용자정의 속성은 모두 소문자 처리되므로, 호출시에도 유의하자.

```
<script>
$(document).ready(function() {
                var userAttr1 = $("#sample_img").data("userattr1");
                var userAttr2 = $("#sample_img").data("userattr2");
                console.log(userAttr1);
                console.log(userAttr2);
});
</script>
```

결과

사용자지정속성값

123456789

#btnModifyDataAttr 버튼을 동적으로 생성하고, 이벤트핸들러를 통해 사용자정의 속성인 userattr1을 변경해 보자.

```
<script>
$(document).ready(function() {
$("<button id='btnModifyDataAttr'>사용자정의 속성변경
</button>").appendTo($("body"));
$("#btnModifyDataAttr").click(function(){
$("#sample_img").data("userattr1", "변경후::사용자정의속성값 ");
var userAttr1 = $("#sample_img").data("userattr1");
                    console.log(userAttr1);
            });
});
</script>
```

결과

변경후::사용자정의속성값

Chapter 5. WEB API

1. Geolocation

지원 브라우저 : 

Geolocation API 는 브라우저가 사용자의 지리적 위치를 찾아내고 그 정보를 애플리케이션에서 이용할 수 있도록 하는 기능이다. 사용자의 위치 정보를 이용하기 위해서는 먼저 승인 절차를 거쳐야 하며, 승인이 완료 된 상태라면 사용자 컨텐츠가 생성될 때 지오-태깅(geo-tagging)기능을 제공할 수 있고 근처에서 촬영된 사진 등에 대한 정보를 유기적으로 연결시켜 서비스할 수 있다.

그리고 사용자의 위치가 변경될 때 마다 콜백 메서드로 전달되어 항상 최신의 위치 정보를 유지하는 것이 가능하다.

이러한 지리 정보는 기본적으로 GPS 장치로 부터 얻어지는 것이 가장 정확하지만 그외 지리정보를 얻을 수 있는 수단들을 단계적으로 이용하여 최소한 수도 또는 국가 단위의 지리 정보를 취득할 수 있다.

다음 예제는 이동 거리 측정기를 만드는 과정을 소개한다.
이 예제를 통해 Geolocation API 의 자세한 사용방법을 알아보자.

브라우저 호환성 확인

geolocation 개체의 존재를 확인하는 방법으로 브라우저가 이를 지원하는지의 여부를 쉽게 확인 할 수 있다.

```
// check for Geolocation support
if (navigator.geolocation) {
  console.log('Geolocation 을 지원합니다.');
}
else {
  console.log('이 브라우저또는 OS 는 Geolocation 을 지원하지 않습니다.');
}
```

측정기의 HTML 마크업

아래는 이동 거리 측정기를 구성하는 HTML 을 마크업 한 것이다.

```
<div id="tripmeter">
<p>
  시작 위치 (위도, 경도):<br/>
<span id="startLat"></span>°, <span id="startLon"></span>°
</p>
<p>
  현재 위치 (위도, 경도):<br/>
  <span id="currentLat"></span>°, <span id="currentLon"></span>°
</p>
<p>
  시작 위치로 부터의 거리:<br/>
  <span id="distance">0</span> km
</p>
</div>
```

사용자의 현재 위치 확인

getCurrentPosition() 메서드를 이용하여 사용자의 현재 위치를 찾아낼 수 있다. 이것은 페이지 로드가 완료되는 시점에 실행된다.

```
window.onload = function() {
  var startPos;
  navigator.geolocation.getCurrentPosition(function(position) {
    startPos = position;
    document.getElementById('startLat').innerHTML = startPos.coords.latitude;
    document.getElementById('startLon').innerHTML =
startPos.coords.longitude;
  });
};
```

만약, 처음으로 위와 같은 과정이 발생하는 경우 브라우저는 사용자에게 위치 정보 사용을 허용할지에 대한 여부를 확인한다.

브라우저에 따라서는 환경설정에서 항상 허용 또는 거부할 수 있는 기능을 제공하기도 하기 때문에 이 과정이 무시될 수도 있다.

위 코드를 실행해 보자.

반환되는 position 개체에서 시작 위치의 좌표를 확인할 수 있어야 한다. position 개체는 위도(latitude)와 경도(longitude) 외에도 많은 정보가 포함되어 있는데, 사용자의 지리정보를 수신하는 기기 환경에 따라서는 고도(altitude)와 방향(direction) 정보까지 얻어낼 수 있다.

console.log 를 이용하여 이 값들의 포함 여부를 살펴보자.

오류 처리

불행하게도 위치 조회를 성공하지 못하는 경우가 발생한다. 어쩌면 갑자기 GPS 를 찾을 수 없거나 위치 정보 사용 권한을 박탈당한 경우일 것이다.

getCurrentPosition() 메서드의 두 번째 인자로 넘긴 콜백은 이러한 오류가 발생한 경우 호출되어 사용자에게 이 사실을 전달할 수 있다.

```
window.onload = function() {
  var startPos;
  navigator.geolocation.getCurrentPosition(function(position) {
  // 상기와 동일
  }, function(error) {
    alert('오류 발생. 오류 코드: ' + error.code);
    // error.code 는 다음을 의미함:
    // 0: 알 수 없는 오류
    // 1: 권한 거부
    // 2: 위치를 사용할 수 없음 (이 오류는 위치 정보 공급자가 응답)
    // 3: 시간 초과
  });
};
```

사용자 위치 모니터링

getCurrentPosition()의 호출은 페이지가 로드되고 난 다음 딱 한 번만 하면 된다. 이후의 위치 변동사항을 추적하려면 watchPosition()을 사용한다. 이것은 사용자의 위치변동이 감지될 때 마다 인자로 받은 콜백을 호출한다.

```
navigator.geolocation.watchPosition(function(position) {
  document.getElementById('currentLat').innerHTML = position.coords.latitude;
  document.getElementById('currentLon').innerHTML =
position.coords.longitude;
});
```

이동한 거리 측정

이 예제는 Geolocation API 와 직접적인 관계는 없다.
하지만 당신이 얻어낸 위치 데이터를 조금 더 구체적으로 이용할 수 있는 방법에 대하여 제시한다.

```
navigator.geolocation.watchPosition(function(position) {
  // 상기와 동일
  document.getElementById('distance').innerHTML =
  calculateDistance(startPos.coords.latitude, startPos.coords.longitude,
  position.coords.latitude, position.coords.longitude);
});
```

지금부터 작성하게 될 calculateDistance() 함수는 두 좌표 사이의 거리를 확인하는 기하학적 알고리즘을 수행한다.

아래의 스크립트 코드는 Moveable Type 에서 제공하는 것으로 Creative Commons 라이선스에 따라 이용할 수 있다.

```
function calculateDistance(lat1, lon1, lat2, lon2) {
 var R = 6371; // km
 var dLat = (lat2 - lat1).toRad();
 var dLon = (lon2 - lon1).toRad();
 var a = Math.sin(dLat / 2) * Math.sin(dLat / 2) +
         Math.cos(lat1.toRad()) * Math.cos(lat2.toRad()) *
         Math.sin(dLon / 2) * Math.sin(dLon / 2);
```

```
  var c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
  var d = R * c;
  return d;
}
Number.prototype.toRad = function() {
  return this * Math.PI / 180;
}
```

이 이동 거리 측정기 예제는 페이지가 로드된 시점으로 부터 일정한 거리를 이동해 봐야 동여부를 확인할 수 있다.

실질적으로 GPS 가 장착된 최신 스마트폰에서 무난히 테스트 할 수 있을 것이다.

데모: https://html5.firejune.com/demo/geolocation.html

위 예제를 http 프로토콜에서 실행하면, 콘솔에서 다음과 같은 에러메세지를 만나게 된다. 어플리케이션을 https 같은 안전한 환경에서 실행해 보자.

```
getCurrentPosition() and watchPosition() no longer work on insecure origins. To use this feature, you should consider switching your application to a secure origin, such as HTTPS. See https://goo.gl/rStTGz for more details.
```

2. Web Storage

지원 브라우저 : 

Web Storage 는 일종의 클라이언트-사이드 데이터베이스이다. 이 데이터는 서버가 아닌 각 사용자의 브라우저에 보관된다. 일반 데이터베이스와의 두드러진 차이점은 우리에게 익숙한 key-value 형식으로 보관/갱신/호출 한다는 것이다. 이것은 Web Storage 를 사용하기위해 별도의 쿼리 문법이나 복잡한 메커니즘을 이해하지 않아도 됨을 의미한다. 그렇기 때문에 우리는 한가지만 기억하면 된다. Web Storage 는 Web Database 와 마찬가지로 브라우저에서 제공하는 저장공간을 사용한다는 것이다. 만약에 사용자가 사파리에서 파이어폭스로 전환하는 경우 동일한 데이터를 가져올 수 없다는 것을 유념하자.

Web Storage 는 localStorage 와 sessionStorage 로 구분된다. 이들의 차이점은 브라우저가 완전히 종료되고 난 후에도 데이터가 유지 되느냐 마느냐이다. 데이터의 용도에 따라서 적절한 방식을 선택하면 된다.

간단한 사용법

자, 이제 간단한 몇 가지 코드를 살펴보자. 다음은 localStorage 의 기본적인 사용법이다.

```
localStorage.setItem("name", "Hello World!"); // key-value 형식으로 저장
document.write(localStorage.getItem("name")); // 저장된 값 호출
localStorage.removeItem("name"); // 스토리지로 부터 일치하는 아이템 삭제
```

첫 번째 라인에서 "name"이라는 키에 "Hello World!"라는 새 항목을 Web Storage 에 저장한 것이다. 여기에서 주의해야 할 점은 setItem 의 두 번째 인자는 항상 문자(String) 형식으로 전달해야 한다.

두 번째 라인에서는 Web Storage 로 부터 "name"키에 저장된 값을 document.write 로 출력한 것이다.

세 번째 라인은 Web Storage 에서 "name"키에 해당하는 데이터를 삭제한 것이다.

만약, 할당량을 초과한 경우 첫 번째 라인에서 오류가 발생하며 데이터가 저장되지않을 것이다.

다음은 이 오류를 대처하는 방법이다.

```
try {
    localStorage.setItem("name", "Hello World!"); // key-value 형식으로 저장
} catch (e) {
    if (e == QUOTA_EXCEEDED_ERR) {
        alert('할당량 초과!'); // 할당량 초과로 인하여 데이터를 저장할 수 없음
    }
}
```

이제 브라우저에서 localStorage 를 지원하지 않는 경우를 구분하자.

```
if (typeof(localStorage) == 'undefined' ) {
  alert('당신의 브라우저는 HTML5 localStorage 를 지원하지 않습니다. 브라우저를 업그레이드하세요.');
} else {
  try {
    localStorage.setItem("name", "Hello World!"); // key-value 형식으로 저장
  } catch (e) {
    if (e == QUOTA_EXCEEDED_ERR) {
      alert('할당량 초과!'); // 할당량 초과로 인하여 데이터를 저장할 수 없음
    }
  }
  document.write(localStorage.getItem("name")); // 저장된 값 호출
  localStorage.removeItem("name"); // 스토리지로 부터 일치하는 아이템 삭제
}
```

이상으로 Web Storage 에 데이터를 저장하고, 호출하고, 삭제하는 간단한 사용법에대하여 알아 보았다.

데모: http://html5.firejune.com/demo/storage.html

쿠키 대신 Web Storage 사용하기

쿠키는 수 년 동안 사용자의 고유 데이터를 추적하는데 사용되어 왔지만 심각한 단점들이 있다. 그 중에도 가장 큰 결함은 모든 쿠키 데이터가 HTTP 요청 헤더에 포함되어 버린다는 점이다. 이는 결국 응답 시간에 나쁜 영향을 미친다. 특히, XHR 이 많은 웹 애플리케이션은 더더욱 그렇다. 가장 좋은 사례는 역시 쿠키의 크기를 줄이는 것이지만 HTML5 에서는 쿠키를 대체할 수 있는 Web Storage 를 사용할 수 있다. localStorage 와 sessionStorage 이 두개의 웹 저장소 개체는 클라이언트-

사이드에 사용자 데이터를 세션이 유지되는 동안 또는 무기한으로 유지하는데 사용할 수 있다. 또한 개인 자료가 HTTP 요청에 전송되지도 않는다. 만약에 사용자 데이터를 쿠키에 저장하고 있다면 다음과 같이 개선해 보자.

```
// 브라우저의 localStorage 지원여부를 판단
if (('localStorage' in window) && window.localStorage !== null){
    // 개체에 프로퍼티를 할당하는 쉬운 방법을 사용
    localStorage.wishlist = '["Unicorn","Narwhal","Deathbear"]';
} else {
    // 브라우저에서 Web Storage 를 지원하지 않는다면
    // doc ument.cookie 를 이용한다.
    var date = new Date();
    date.setTime(date.getTime()+(365*24*60*60*1000));
    var expires = date.toGMTString();
    var cookiestr = 'wishlist=["Unicorn","Narwhal","Deathbear"];'+
                    ' expires='+expires+'; path=/';
    document.cookie = cookiestr; }
```

3. Web SQL Database

지원 브라우저 : 

Web Database 는 HTML5 와 함께 새로 생겨난 것이다. 이제부터 클라이언트 웹 개발자들은 풍부한 쿼리 능력을 가진 웹 애플리케이션을 만들 수 있게 되었다. SQL 쿼리를 별도로 익혀야하는 노고가 뒤따르지만 온라인 또는 오프라인 여부에 상관없이 사용 가능하며, 클라이언트의 저장소에 영구히 보존할 수 있고, 리소스 점유율이 많은 덩치큰 데이터를 체계적으로 관리 할 수 있다.

지금 소개할 예제 코드들은 아주 간단한 할 일 목록 관리 애플리케이션을 만드는 과정을 다룬다.

변수 선언

예제에 사용될 데이터베이스 로직은 아래와 같은 네임스페이스를 사용한다.

```
var html5rocks = {};
html5rocks.webdb = {};
```

비동기와 트랜잭션의 이해 Web Database 를 사용하는 대부분의 사례가 비동기 API 를 사용한다. 비동기 API 는 non-blocking 시스템이다. 그리고 리턴값을 통해서는 데이타를 얻지 못한다. 때문에 정의된 콜백 함수에 데이터를 전달하게 된다.

Web Database 는 HTML 을 통한 트랜잭션이다. 이것은 외부에서 SQL 문을 실행할 수 없다. 트랜잭션은 두 종류로 구분되는데, 읽고 쓰기위한 트랜잭션(transaction())과 읽기 전용 트랜잭션(readTransaction())이다.

그리고 주의해야 할 점은 데이터를 읽고 쓸 때 전체 데이터베이스가 잠겨버린다는 점이다.

데이터베이스 열기

데이터베이스에 접근하기 전에 먼저 해야할 일은 데이터베이스를 개설하는 것이다. 개설하기 위해서는 데이터베이스의 이름, 버전, 설명 그리고 크기를 정의 한다.

```
html5rocks.webdb.db = null;
html5rocks.webdb.open = function() {
    var dbSize = 5 * 1024 * 1024; // 5MB
```

```
        html5rocks.webdb.db = openDatabase('Todo', '1.0', 'todo manager', dbSize);
    }
    html5rocks.webdb.onError = function(tx, e) {
        alert('예기치 않은 오류가 발생하였습니다: ' + e.message );
    }
    html5rocks.webdb.onSuccess = function(tx, r) {
        / / 모든 데이터를 다시 그림
        html5rocks.webdb.getAllTodoItems(tx, r);
    }
```

테이블 생성하기

"CREATE TABLE SQL" 쿼리문을 transaction 안에 실행하여 테이블을 만들 수 있다.
OnLoad 이벤트가 발생하는 지점에 테이블 생성함수를 정의했다.
테이블이 존재하지 않는 경우에는 테이블이 생성된다. 이 테이블의 이름은 "todo"이고 아래와 같은 3 개의 컬럼을 가진다.

• ID - 순차적으로 증가하는 ID 컬럼
• todo - 아이템의 몸체가 되는 텍스트 컬럼
• added_on - 아이템이 만들어진 시간 컬럼

```
    html5rocks.webdb.createTable = function(){
    html5rocks.webdb.db.transaction(function(tx) {
        tx.executeSql('CREATE TABLE IF NOT EXISTS ' +
                      'todo(ID INTEGER PRIMARY KEY ASC, todo TEXT, added_on
    DATETIME)', []);
        });
    }
```

테이블에 데이터 추가하기

할 일 목록을 관리하기 위한 테이블이 준비되었다. 이제 테이블에 아이템을 추가하는 중요한 작업을 진행해 보자.

transaction 내부에서 todo 테이블에 INSERT 쿼리를 수행해야 한다.

이 때 executeSql 은 다수의 파라미터를 가진다. 그리고 SQL 은 이 파라미터의 값을 컬럼에 입력하는 쿼리를 수행한다.

```
html5rocks.webdb.addTodo = function(todoText) {
    html5rocks.webdb.db.transaction(function(tx){
    tx.executeSql('INSERT INTO todo(todo, added_on) VALUES (?,?)',
        [todoText, addedOn],
        html5rocks.webdb.onSuccess,
        html5rocks.webdb.onError);
    });
}
```

테이블에서 데이터 선택하기

이제 데이터베이스에 데이터가 존재한다. 이 데이터를 다시 밖으로 꺼내보자. Web Database 는 표준 SQLite SELECT 쿼리를 이용하면 된다.

```
html5rocks.webdb.getAllTodoItems = function(renderFunc) {
    html5rocks.webdb.db.transaction(function(tx) {
        tx.executeSql('SELECT * FROM todo',
                    [],
                    renderFunc,
                    html5rocks.webdb.onError);
        });
}
```

여기에 사용된 명령 예제는 모두 비동기이다. 이러한 경우 transaction 또는 executeSql 호출시 데이터가 반환되지 않는다. 데이터는 반드시 콜백을 통해 전달된다는 사실을 기억하자.

가져온 데이터 처리하기

데이터를 성공적으로 가져왔다면 loadTodoItems 함수가 호출되게 하자. onSuccess 콜백은 두개의 파라미터를 가진다. 첫 번째는 쿼리 트랜잭션이고 두 번째는 결과 묶음이다. 결과 묶음은 배열이며 데이터가 담겨 있다.

```
function loadTodoItems(tx, rs) {
    var rowOutput = "";
    for (var i=0; i < rs.rows.length; i++) {
        rowOutput += renderTodo(rs.rows.item(i));
    }
```

```
        var todoItems = document.getElementById('todoItems');
        todoItems.innerHTML = rowOutput;
    }
    function renderTodo(row) {
        return '<li>' + row.ID +
        '[<a onclick="html5rocks.webdb.deleteTodo(' + row.ID + ');"'>X</a>]</li>';
    }
```

그리고 ID 가 "todoItems"인 DOM 요소에 할 일 목록들이 그려지는 일을 수행한다.

테이블에서 데이터 제거하기

```
    html5rocks.webdb.deleteTodo = function(id) {
        html5rocks.webdb.db.transaction(function(tx) {
        tx.executeSql('DELETE FROM todo WHERE ID=?', [id],
                        loadTodoItems, html5rocks.webdb.onError);
        });
    }
```

초기화 및 HTML 구성하기

페이지 로드가 완료되면, 데이터베이스를 열고, 테이블을 생성하고(필요한 경우), 데이터를 가져와 할 일 항목이 그려지게 하자.

```
    <script>
    ....
    function init() {
        html5rocks.webdb.open();
        html5rocks.webdb.createTable();
        html5rocks.webdb.getAllTodoItems(loadTodoItems);
    }
    </script>
    <body onload="init();">
      <form type="post" onsubmit="addTodo(); return false;">
        <input type="text" id="todo" name="todo"
        placeholder="What do you need to do?" style="width: 200px;" />
        <input type="submit" value="Add Todo Item"/>
      </form>
```

<input> 요소로 부터 작성된 값을 가져와 전달하기 위한 함수가 필요하다.
html5rocks.webdb.addTodo 메서드를 호출할 함수를 만들자.

```
function addTodo() {
    var todo = document.getElementById('todo');
    html5rocks.webdb.addTodo(todo.value);
    todo.value = '';
}
```

데모 : http://html5.firejune.com/demo/webdb.html

4. Drag and Drop

지원 브라우저 : 

Drag and Drop API 가 없던 시절에도 "mousemove", "mousedown", "mouseup" 이벤트를 이용하여 요소를 특정한 요소에 끌어다 놓는 수준은 구현할 수 있었다. 그러나 잡다한 뒤처리를 해야 했기 때문에 자바스크립트 라이브러리를 추가적으로 이용해야 했고 이벤트 이상 증식현상이나 CPU 부하로 인한 오작동이 빈번하게 발생하여 널리 사용되고 있지는 않았다.

HTML5 에서 새롭게 지원하기 시작한 Drag and Drop API 는 더욱 향상된 끌어다 놓기 경험을 제공한다.

특히, File API 를 함께 이용하면, 바탕화면 혹은 탐색기의 파일을 브라우저로 직접 끌어나 놓는 방식으로도 파일을 업로드 할 수 있게 되었다.

이 예제는 로컬에 위치한 파일을 특정한 HTML 요소에 끌어다 놓고 해당 파일을 직접 액세스하고 미리보기를 보여주는 예제이다.

드랍 영역 마크업하기

아이템을 드래그할 수 있는 영역과 미리볼 수 있는 이미지를 등록한다.

```
<div id="dropbox">
<span id="droplabel">
이곳에 파일을 드랍해 주세요...
</span>
</div>
<img id="preview" alt="[ preview will display here ]" />
```

드랍 영역 이벤트 등록하기

그리고 아래와 같이 이벤트를 할당한다.

```
var dropbox = document.getElementById("dropbox")
// 이벤트 핸들러 할당
dropbox.addEventListener("dragenter", dragEnter, false);
dropbox.addEventListener("dragexit", dragExit, false);
dropbox.addEventListener("dragover", dragOver, false);
dropbox.addEventListener("drop", drop, false);
```

위 코드는 얼핏 보면 복잡해 보이지만 dragEnter, dragExit, dragOver 핸들러는 아래와 같은 이벤트의 이상 증식현상을 중지시키는 역할을 할 뿐이다.

```
event.stopPropagation();
event.preventDefault();
```

drop 이벤트 헨들러 작성하기

반환된 이벤트로 부터 dataTransfer.files 개체로 접근한 후 파일이 1 개 이상 존재하면 handleFiles 함수를 호출한다.

```
event.stopPropagation();
event.preventDefault();
var files = event.dataTransfer.files;
var count = files.length;
// 오직 한개 이상의 파일이 드랍된 경우에만 처리기를 호출한다.
if (count > 0)
handleFiles(files);
```

handleFiles 함수 작성하기 전달 받은 개체로 부터 파일들 선택하고, 파일 이름을 표시하고, FileReader(File API) 인스턴스를 생성하여 파일을 처리한다.

```
var file = files[0];
document.getElementById("droplabel").innerHTML = "Processing " + file.name;
var reader = new FileReader(); // 파일 리더의 이베트 핸들러 정의
reader.onloadend = handleReaderLoadEnd; // 파일을 읽는 작업 시작
reader.readAsDataURL(file);
```

readAsDataURL 메서드는 파일을 data URL 형식으로 만들어 준다. 이는 파일을 서버에 업로드하지 않고도 조작할 수 있음을 의미한다. 포멧을 변환하거나, 데이터를 분석하여 변조하는 일이 가능해 진다.
예를 들면, 이미지의 특정한 영역을 클라이언트-사이드에서 크롭한 후 서버에 업로드하는 것이 가능하다.

handleReaderLoadEnd 함수 작성하기

handleReaderLoadEnd 함수의 내용은 아주 간단하다. 미리보기할 이미지 요소에 소스를 대입한다.

```
var img = document.getElementById("preview"); img.src = event.target.result;
```

데모: http://html5.firejune.com/demo/dnd.html

5. Web Workers

지원 브라우저 : 

Web Worker 는 두가지 중요한 장점을 가지고 있다. 첫번째는 빠르다는 것이고, 두번째는 브라우저에 부담을 주지않고 백그라운드에서 스크립트 연산을 수행하는 것이다. 이것이 가능한 이유는 브라우저가 OS-레벨의 스레드를 생성하기 때문이며, 동시 다발적으로 사용하는 경우 더욱 흥미로운 결과를 기대할 수 있다. 이제부터 Web Worker 기본적인 사용법에 대하여 알아보자.

Worker 생성하기

Worker 를 생성하는 것은 간단하다. 백그라운드에서 작업할 스크립트를 별도의 파일에 작성하고 새로운 인스턴스에 URI 를 기입하여 생성한다. 그리고 onmessage 속성에 함수를 대입하여 작업결과를 돌려 받을 수 있다.

```
var myWorker = new Worker('my_worker.js');
myWorker.onmessage = function(event) {
    alert("Worker 에 의해 실행된 콜백!₩n");
};
```

Worker 종료하기 실행중인 Worker 를 즉시 종료하려면 terminate() 메서드를 호출하여 즉시 종료할 수 있다. 이러한 경우, Worker 는 남은 작업을 마무리하거나 메모리에서 찌꺼기를 청소한 후 자발적으로 사라진다.

```
myWorker.terminate();
```

백그라운드에서 피보나치 수열 계산하기

다음 예제는 Worker 를 이용하여 피보나치 수열을 계산하는데 사용된다. 이것은 사용자 인터페이스의 스레드를 차단하지 않고 프로세서 집약적인 계산을 수행 할 수 있도록 하는 것이다.

다음은 "fibonacci.js"에 저장된 내용이다.

```
var results = [];
function resultReceiver(event) {
  results.push(parseInt(event.data));
```

```
    if (results.length == 2) {
      postMessage(results[0] + results[1]);
    }
  }
  function errorReceiver(event) {
    throw event.data;
  }
  onmessage = function(event) {
    var n = parseInt(event.data);
    if (n == 0 || n == 1) {
      postMessage(n);
      return;
    }
    for (var i = 1; i <= 2; i++) {
      var worker = new Worker("fibonacci.js");
      worker.onmessage = resultReceiver;
      worker.onerror = errorReceiver;
      worker.postMessage(n - i);
    }
  };
```

onmessage 함수는 postMessage()를 호출한다. 이렇게 함으로써 반복적인 계산의 새로운 복사본을 만들어 수행하게 된다

```
  <!DOCTYPE html>
  <html>
  <title>Test threads fibonacci</title>
  <body>
    <div id="result"></div>
    <script language="javascript">
      var worker = new Worker("fibonacci.js");
      worker.onmessage = function(event) {
      document.getElementById("result").textContent = event.data;
      console.log("Got: " + event.data + "₩n");
      };
      worker.onerror = function(error) {
```

```
        console.log("Worker error: " + error.message + "₩n");
            throw error;
        };
        worker.postMessage("5");
    </script>
</body>
</html>
```

id 가 "result"인 요소에 그 결과가 표시되며 worker.postMessage 에 의해 Worker 에 작업을 지시할 수 있다.

데모: http://html5.firejune.com/demo/worker.html

CPU 부하를 줄이기 위한 Web Worker 를 적용하기에 적합한 상황들

스크립트를 이용하여 무거운 연산을 실행하면 브라우저는 먹통(응답 없음) 상태가 된다. 이러한 경우 이벤트 리스너가 제대로 작동하지 않아 오작동이 발생하거나 제때 콜백이 호출되지 않거나 짧은 시간동안 상호작용이 발생하는 프로그램 로직에 치명적인 오류를 안겨줄 수 있다.

이러한 상황은 Web Worker 를 이용하여 우회할 수 있다는 사실을 기억하자.

- 긴 문서의 문자 서식 지정
- 문법 강조 기능
- 이미치 프로세싱
- 이미지 합성
- 덩치큰 배열 처리

6. Web Sockets

지원 브라우저 : 

Web Socket 은 꾸준한 성장과 인기를 얻고있는 Comet 의 대안으로 고안되었다. 이것은 웹 애플리케이션이 full-duplex 단일 소켓 연결을 가능케 한다. 이는 서버와 브라우저 사이에 진정한 양방향 통신 채널을 제공하는 것을 의미하며, 연결 관리를 단순화 한다. 하지만 서버에서 Web Sockets 프로토콜을 지원하는 환경에서만 작동하며, 추가적으로 서버에 모듈을 설치하거나 독립적으로 이를 지원하는 서버에서 정상적으로 작동한다.

XHR 보다 적은 대역폭을 가진 빠른 송/수신

WebSocket 은 매우 가볍게 구성되어 XHR 보다 대역폭 소모가 적다. 일부 보고서에 따르면 전송 대역폭의 35%의 절감효과가 발생하는 것으로 조사되었다. 또한, 메시지 전달 비교 실험에서 XHR 이 WebSocket 보다 3500% 느린 것으로 측정됨으로써 상당 수준의 성능 차이가 있는 것으로 밝혀졌다.

끝으로, Ericcson 연구소에서 만든 WebSockets 와 HTTP 비교 동영상은 WebSockets 보다 HTTP 가 ping 당 3-5 배나 느린것으로 밝혀져 실시간 상호작용이 빈번하게 발생하는 웹 애플리케이션 개발에 사용하기 적합한 것으로 결론 내렸다.

지금부터 서버와 클라이언트간 메시지를 주고 받는 간단한 예제를 살펴보자.

클라이언트-사이드

클라이언트-사이드의 Web Socket 은 매우 간단하게 사용할 수 있도록 고안되었다. 다음 코드가 하는 일은 서버의 9876 포트에 접속하고 수신한 데이터를 alert 으로 출력한다.

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Web Socket Example</title>
<meta charset="UTF-8">
<script>
  window.onload = function() {
    var s = new WebSocket("ws://localhost:9876/");
    s.onopen = function(e) { alert("opened"); }
```

```
        s.onclose = function(e) { alert("closed"); }
        s.onmessage = function(e) { alert("got: " + e.data); }
    };
    </script>
    </head>
    <body>
      <div id="holder" style="width:600px; height:300px"></div>
    </body>
    </html>
```

서버-사이드

이제 서버 차례다. 서버는 3 초 간격으로 두개의 메시지를 보낸다. 단순성과 명확성을 위해 서버측 응답은 hard-coding 된 것으로 한다. 이를 실제로 구현 한다면 유효성을 검사하고 동적인 응답이 이루어지도록 해야할 것이다.

myServerSocket.java

```
import java.io.*;
import java.net.*;
import java.util.*;

import javax.swing.plaf.SliderUI;

import org.java_websocket.*;
import org.java_websocket.handshake.ClientHandshake;
import org.java_websocket.server.WebSocketServer;



public class myServerSocket extends WebSocketServer {
        int cnt; //접속자
        boolean startFlag = false;
        StringBuffer sb = new StringBuffer();

        //생성자 2
        public myServerSocket( int port ) throws UnknownHostException {
```

```
                super( new InetSocketAddress( port ) );
        }

        //생성자 1
        public myServerSocket( InetSocketAddress address ) {
                super( address );
        }

        @Override
        public void onOpen( WebSocket conn, ClientHandshake handshake ) {
                this.sendToAll( "new connection: " +
conn.getRemoteSocketAddress().getAddress().getHostAddress());
                this.sendToAll("startFlag: "+startFlag);
                //this.sendToAll( "new connection: " +
handshake.getResourceDescriptor() );

        System.out.println(  conn.getRemoteSocketAddress().getAddress().getHostA
ddress() + " entered the room!" );
                cnt++;
                System.out.println("현재접속자::::::"+cnt);
        }

        @Override
        public void onClose( WebSocket conn, int code, String reason, boolean
remote ) {
                this.sendToAll( conn + " has left the room!" );
                System.out.println( conn + " has left the room!" );
                cnt--;
                System.out.println("현재접속자::::::"+cnt);
        }

        @Override
        public void onMessage( WebSocket conn, String message ) {
        }
```

```
	public static void main( String[] args ) throws InterruptedException , IOException {
		WebSocketImpl.DEBUG = true;
		int port = 9999; // 843 flash policy port
		try {
			port = Integer.parseInt( args[ 0 ] );
		} catch ( Exception ex ) {

		}
		myServerSocket s = new myServerSocket( port );
		s.start();
		System.out.println( ">> myChatServer started on port: " + s.getPort() );

		//메세지보내기 쓰레드 생성 및 스타트!
		SendMessage serverMessenger = new SendMessage("serverMessenger", s);
		serverMessenger.start();

		//서버에서 클라이언트에게 보내는 채팅
		BufferedReader sysin = new BufferedReader( new InputStreamReader( System.in ) );

		while ( true ) {
			String in = sysin.readLine();
			System.out.println(">>Server Says => "+in);
			s.sendToAll( in );
		}
	}

	@Override
	public void onError( WebSocket conn, Exception ex ) {
		ex.printStackTrace();
```

```
                    if( conn != null ) {
                            // some errors like port binding failed may not be
assignable to a specific websocket
                    }
            }
            /**
             * Sends <var>text</var> to all currently connected WebSocket clients.
             *
             * @param text
             *                  The String to send across the network.
             * @throws InterruptedException
             *                  When socket related I/O errors occur.
             */
            public void sendToAll( String text ) {
                    Collection<WebSocket> con = connections();
                    synchronized ( con ) {
                            for( WebSocket c : con ) {
                                    System.out.println(text);
                                    c.send( text );
                            }
                    }
            }
}
```

sendMessage.java

```
    public class SendMessage extends Thread {
        private boolean msgFlag=true;
        private String name;
        private myServerSocket s;

        public SendMessage(String name, myServerSocket s){
                this.name = name;
                this.s = s;
```

```
        }
        @Override
        public void run(){
                for (int i = 0; i < 10; i++) {
                        try {
                                sleep(3000);
                                String msg = msgFlag?"hello ":"world ";
                                s.sendToAll(name+"::"+msg);

                                msgFlag = !msgFlag;

                        } catch (InterruptedException e) {
                                e.printStackTrace();
                        }
                }
        }
}
```

톰캣서버를 작동하고, 채팅서버를 구동(Java Application)하면, "hello"와 "world"라는 메시지가 3 초간격으로 수신되는 모습을 확인할 수 있다.

```
open!!
message::new connection: 0:0:0:0:0:0:0:1
message::startFlag: false
message::new connection: /
message::serverMessenger::hello
message::serverMessenger::world
message::serverMessenger::hello
message::serverMessenger::world
message::serverMessenger::hello
message::serverMessenger::world
message::serverMessenger::hello
message::serverMessenger::world
message::serverMessenger::hello
message::serverMessenger::world
```

7. WebGraphics

Canvas 요소

지원 브라우저 :

사실, <canvas>요소는 HTML5 가 나오기 전부터 존재했었고 다양한 용도로 사용되어 왔었지만 이제서야 HTML5 의 공식 명세로 자리잡았다. 이 요소를 사용하면 2 차원의 비트맵 이미지 프로세싱이 가능하고 동적인 그래픽 렌더링을 스크립트로 제어할 수 있다. 이는 웹 페이지에 인터랙티브한 그래픽 컨텐츠를 만들어 제공할 수 있음을 뜻한다. 화려한 그래픽 기반의 개임이나, 다양한 종류의 그래프, 이미지 합성 또는 변형, 드로잉 애플리케이션 등 마치 플래시로 생성된 것과 같은 컨텐츠를 제공할 수 있게 되는 것이다.

간단한 예제

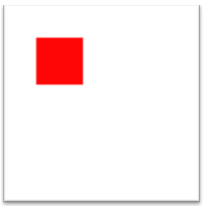
<canvas>요소에 빨간색 사각형을 그려보자. 아래 코드는 HTML 문서에 <canvas>요소를 작성한 것이다.

```
<canvas id="example" width="200" height="200">
이 메시지는 사용자의 브라우저에서 HTML5 캔버스를 지원하지 않는 경우 표시 됨
</canvas>
```

스크립트를 사용하여 <canvas>요소에 사각형을 그려 넣는다.

```
var example = document.getElementById('example');
var context = example.getContext('2d');
context.fillStyle = "rgb(255,0,0)";
context.fillRect(30, 30, 50, 50);
```

다음과 같은 결과를 얻을 수 있다.



데모 : http://html5.firejune.com/demo/canvas.html

8. SVG 요소

지원 브라우저 : 

HTML5 명세에 포함되면서부터 표준으로 자리매김한 SVG 는 확장 가능한 백터 그래픽(Scalable Vector Graphics)의 줄임말이다.

2 차원 백터 그래픽만을 표현하며, XML 형식으로 작성되고, SVG 뷰어를 이용하는 등 다양한 삽입 방법으로 사용자가 조회할 수 있다. SVG 의 작성은 HTML 과 매우 유사하다. <circle>,<rect> 등과 같은 그래픽 태그들을 이용하여 작성하면 된다.

SMIL 또는 스크립트를 이용하여 동적인 변화를 주거나 CSS 를 지정하여 모양을 꾸밀 수도 있다. SVG 와 스크립트를 접목하여 상호작용이 발생하는 차트, 다이어그램, 일러스트레이트 등 선명한 화질을 가진 확대 가능한 자료를 웹 페이지에 삽입할 수 있으며, 마인드맵, 목업과 같은 애플리케이션을 개발할 수 있다. SVG 요소의 마크업 보통 아래와 같은 형식으로 HTML 문서에 마크업 한다.

```
<svg xmlns="http://www.w3.org/2000/svg"
      xmlns:xlink="http://www.w3.org/1999/xlink">
     <rect x="10" y="10" height="100" width="100"
          style="stroke:#ff0000; fill: #0000ff"/>
</svg>
```

또는 리소스를 지정하는 형식으로도 삽입할 수 있다. 경우에 따라서는 svg 뷰어 브라우저 플러그인을 필요로 하기도 한다

```
<!-- object 요소 사용 -->
<object data="/svg/examples.svg" width="300" height="100" type="image/svg+xml"
codebase="http://www.adobe.com/svg/viewer/install/" />
<!-- embed 요소 사용 -->
<embed src="/svg/examples.svg" width="500" height="200" type="image/svg+xml"
pluginspage="http://www.adobe.com/svg/viewer/install/" />
<!-- iframe 요소 사용 -->
<iframe src="/svg/examples.svg" width="300" height="100"></iframe>
```

삽입 결과는 다음과 같다.

데모 : http://html5.firejune.com/demo/svg.htm

SVG 와 스크립트

SVG 에 스크립트로 애니메이션을 하거나 변화를 주는 일은 DOM 을 스크립트로 다루는 것과 별반 다르지 않다. 다음은 SVG 가 가진 특정한 요소를 스크립트와 SMIL 을 이용하여 애니메이션하는 예제이다.

```
var svgDocument;
var svgns = 'http://www.w3.org/2000/svg';
var xlinkns = 'http://www.w3.org/1999/xlink';
function startup(evt){
	P=document.getElementById("P")
	CL=document.getElementById("CL")
	animate()
	stop("S")
	stop("L")
}
limit=720
blu=4
speed=6
running=true
function animate(){
	if (!running) return

	B="rotate("+blu+" 360 150)"
	C="rotate("+(-blu/2)+" 360 150)"
	CL.setAttribute("transform", B);
```

```
            P.setAttribute ("transform", C);
            blu=blu+speed
            if ((blulimit)) speed=-speed
            window.setTimeout("animate()",10)
        }
        runAnim=new Object
        runAnim["S"]=false
        runAnim["L"]=false
        function stop(id){
            if (runAnim[id]){
                document.getElementById(id).firstChild.nextSibling.endElement()
                document.getElementById("E"+id).endElement() }
            else{
                document.getElementById(id).firstChild.nextSibling.beginElement()
                document.getElementById("E"+id).beginElement()
            }
            runAnim[id]=!runAnim[id]
}
```

데모 : http://html5.firejune.com/demo/svg-script.svg