



# (스마트웹&콘텐츠 개발) 스마트웹&콘텐츠 응용SW 개발자 양성 과정

HTML5, CSS3, Javascript,  
jQuery를 활용한 UI 디자인 및  
반응형 웹 구현





스마트웹&콘텐츠 개발

HTML5, CSS3, Javascript, jquery를 활용한 UI 디자인 및 반응형 웹 구현

KH정보교육원







# (스마트웹&콘텐츠 개발) 스마트웹&콘텐츠 응용SW 개발자 양성 과정

HTML5, CSS3, Javascript,  
jQuery를 활용한 UI 디자인 및  
반응형 웹 구현





**KH정보교육원**

(스마트웹&콘텐츠 개발) 스마트웹&콘텐츠 응용SW 개발자 양성 과정

HTML5, CSS3, Javascript, jQuery를 활용한 UI 디자인 및  
반응형 웹 구현

CONTENTS

Part 1. 인터넷과 웹	
Chapter 1. 인터넷과 웹 서비스의 이해	4
Chapter 2. 웹 표준이란?	6
Chapter 3. 웹 표준의 장점	8
Chapter 4. 웹 호환성	10
Part 2. 반응형 웹 구현	
Chapter 1. HTML을 활용한 기본 웹페이지 구성	14
Chapter 2. CSS3을 활용한 웹 디자인 및 스타일 꾸미기	46
Chapter 3. 다양한 기기에 반응하는 반응형 웹 제작	88
Chapter 4. JavaScript를 활용하여 다양한 기능 추가하기	102
Chapter 5. jQuery	189
Chapter 6. WEB API 활용	237





## **Part 1. 인터넷과 웹**

## **Chapter 1. 인터넷과 웹 서비스의 이해**

## 1. 인터넷이란?

international network 의 약자로 처음 시작은 미국의 국방부의 군사통신망인 알파넷(ARPA(The Advanced Research Project Agency)-net 이 시작입니다.

인터넷(internet)의 사전적 정의를 보면, 아르파넷(ARPANET)에서 시작된 세계 최대 규모의 컴퓨터 통신망으로, 통신망과 통신망을 연동해 놓은 망의 집합을 의미하는 인터넷워크(internetwork)의 약어인 internet 과 구별하기 위해 Internet 또는 INTERNET 과 같이 고유명사로 표기합니다. 랜(LAN) 등 소규모 통신망을 상호 접속하는 형태에서 점차 발전하여 현재는 전세계를 망라하는 거대한 통신망의 집합체가 되었습니다.

인터넷은 한마디로 지금 사용하는 것처럼 온라인이 연결되어있는 곳에서 누구나 접속해서 온라인상의 정보/지식 등을 게시 및 공유하고 서로 교환할 수 있도록 해주는 통신망을 가리킵니다

## 2. 웹 서비스의 이해

웹 서비스(Web Service)는 네트워크 상에서 서로 다른 종류의 컴퓨터들 간에 상호작용을 하기 위한 소프트웨어 시스템이다. 웹 서비스는 서비스 지향적 분산 컴퓨팅 기술의 일종으로 SOAP, WSDL, UDDI 등의 주요 표준 기술로 이루어진다. 웹 서비스의 모든 메시징에는 주로 XML 이 사용된다. -Wikipedia-

웹 서비스의 구성도를 도식화 하면 아래 그림과 같다.



## **Chapter 2. 웹 표준이란?**

## 1. 웹 표준이란?

웹 표준이란 브라우저 종류 및 버전에 따른 기능 차이에 대하여 호환이 가능하도록 제시된 표준으로, 다른 기종 혹은 플랫폼에 따라 달리 구현되는 기술을 동일하게 구현함과 동시에 어느 한쪽에 최적화되어 치우치지 않도록 공통요소를 사용하여 웹 페이지를 제작하는 기법을 의미한다. 표준화 단체인 W3C(World Wide Consortium)가 권고한 표준안에 따라 웹사이트를 작성할 때 이용하는 HTML, CSS, JavaScript 등에 대한 규정을 담고 있으며 웹 표준의 궁극적인 목적은 웹사이트에 접속한 사용자가 어떠한 운영체제나 브라우저를 사용하더라도 동일한 결과를 보이게 하는 것이다.

국내 웹의 현실은 특정 브라우저와 사용자 등의 이용환경과 비표준 페이지 및 과도한 플러그인 사용으로 장애인, 노약자를 포함한 모든 사용자들에게 운영체제 및 웹 브라우저 등의 정보 접근 제약이 있다. 따라서 브라우저의 종류나 버전에 상관없이 모든 사용자들이 동일한 웹사이트를 볼 수 있도록 웹 표준기술 작업이 필요하며 웹 표준 준수는 웹 접근성 준수를 위한 핵심이다

.웹 표준을 논할 때 일반적으로 다음의 것들이 중요성이 있는 것으로 보인다:

- HTML, XHTML, SVG, XForms 와 같은 마크업 언어의 W3C 권고
- 스타일시트, 특히 CSS 의 W3C 권고
- 흔히 자바스크립트나 ECMA 스크립트로 불리는 Ecma 인터내셔널 표준
- 문서 객체 모델의 W3C 권고

웹 접근성은 일반적으로 W3C 의 Web Accessibility Initiative 가 출판한 웹 콘텐츠 접근성 가이드라인\*에 기반을 두고 있다.

\* <https://www.w3.org/TR/WAI-WEBCONTENT/>

## **Chapter 3. 웹 표준의 장점**

## 2. 웹 표준의 장점

### 수정 및 운영관리 용이

콘텐츠의 올바른 구조화와 CSS 로 시각표현을 통일하여 제어하게 되어 페이지 제작의 부담 감소 및 관리용이

### 접근성 향상

웹 표준을 이용해 작성한 문서는 다양한 브라우징 환경에 대응이 가능하며 휴대폰, PDA 에서도 정상적인 작동 및 장애인 지원용 프로그램에도 도움이 되므로 사용자나 접속 장치의 접근성이 용이

### 검색엔진 최적화(SEO)

구조화된 웹페이지는 검색 로봇 수집을 통해 검색엔진에 효율적으로 노출 될 수 있도록 검색엔진의 검색결과를 최적화

### File Size 축소, 서버 저장 공간 절약

효율적인 소스 작성은 파일 사이즈와 서버공간을 절약할 수 있으며 동시에 화면표시에 소요되는 시간을 단축

### 효율적인 마크업

CSS 와 HTML 문서를 분리하여 제작할 경우 불필요한 마크업이 최소화되어 페이지 로딩속도 향상

### 호환성 가능

기존 IE 브라우저에서만 작동이 가능했던 요소들이 웹 표준을 준수함으로써 다양한 브라우저 (크롬, 파이어폭스, 사파리, 오페라 등)에서도 작동

## **Chapter 4. 웹 호환성**



### 3. 웹 호환성이란?

웹 호환성은 표준 웹 기술을 사용하여 운영체제, 브라우저 등 어느 한쪽으로 최적화되거나 종속되지 않도록 공통 요소를 사용하여 웹 페이지를 제작하는 기법으로 웹 사이트 사용 시 운영체제 및 브라우저 간 동일한 결과가 나오도록 의미하는 웹 상호운용성의 개념에 웹 표준의 준수를 포함하는 개념이다.

### 4. 웹 호환성을 준수해야 하는 이유

웹 표준을 준수하지 않고 특정 운영체제와 브라우저에 종속되어 있다면 다양한 운영체제 환경 및 브라우저에서의 사용이 불가능한 문제점이 발생하게 된다.

국내에는 Internet Explorer 웹 브라우저에 최적화된 비표준 기술이 널리 사용되고 있다. 개방형 통합 플랫폼인 ActiveX 가 그 대표적인 예로, ActiveX 는 웹사이트에서 정적인 웹문서를 멀티미디어 기술로 동작 가능하게 하는 플러그인(Plug-in) 기술이나 보안에 취약한 문제점이 있으며 또한 IE 에서만 사용되는 기술로 다른 웹 브라우저에서는 구동이 불가능하여 외국에서 국내 웹 사이트에 접근할 때 표준화되지 않은 웹사이트로 인해 웹 호환성이 현격히 떨어지는 문제가 발생할 수 있다. 따라서 제공하는 서비스를 모든 웹 브라우저 환경에서 동일하게 이용하기 위해서는 웹 호환성을 준수한 웹사이트의 구축이 필요하다.

### 5. 웹표준, 웹호환성, 웹접근성의 이해

웹 표준, 웹 호환성, 웹 접근성의 목적은 웹을 사용하는 사용자가 웹사이트를 자유롭고 편리하게 이용하는 점과 장애인, 고령자 등을 포함한 사용자층 확대, 다양한 환경, 새로운 기기에서의 이용, 개발 및 운용의 효율성 제고 등의 기대효과가 유사하지만 대상 및 종류 등의 준수 내용과 편의를 제공하는 점에서 차이가 있다. 웹 표준을 준수 하는 것만으로는 웹 접근성이나 웹 호환성이 보장되지 않으며 웹 호환성을 준수하더라도 웹 접근성은 보장되지 않는다. 웹 접근성이란 보편적 접근성 확보를 우선시하고, 웹 호환성은 OS, SW 에 독립적인 상호운용성 확보를 우선시한다.



구분	목적	준수 내용	차이
웹 표준 (Web Standards)	웹의 사용성 및 접근성 보장	HTML, CSS 등에 대한 WC3규격(문법) 준수 등 - HTML, CSS, Javascript 등 구조와 표현, 동작 분리 권고	웹의 내용, 표현, 행동 에 관련된 기술표준
웹 호환성 (Cross Browsing)	웹 브라우저 버전, 종류와 관계없는 웹사이트 접근	웹 표준 준수를 통한 브라우저 호환성 확보 - HTML, CSS 문법 준수 - 동작, 레이아웃, 플러그인 호환성	웹 표준을 공통으로 포함
웹 접근성 (Web Accessibility)	인적, 환경적 요인에 제약없는 웹 정보 접근	W3C 웹 접근성 이니셔티브(WAI) 한국형 웹 콘텐츠 접근성 지침2.0 - 인식의 용이성, 운용의 용이성, 이해의 용이성, 견고성	

## **Part 2. 반응형 웹 구현**

# **Chapter 1. HTML을 활용한 기본 웹페이지 구성**

# 1. HTML이란

## HTML

HyperText Markup Language의 약자로 웹의 최소 단위인 웹 페이지를 만드는 언어이다. 태그로 되어있는 HTML요소 형태로 웹 페이지 구조를 표현한다

## 웹표준 - HTML5

웹표준이란 다양한 접속환경을 가진 인터넷 사용자들이 정보에 소외되지 않고, 모두가 동등하게 정보를 이용할 수 있도록 하기 위해 표준에 따라 웹을 개발하는 것을 말한다. 이는 웹 접근성의 증대로 이어진다. 웹접근성이란 누구든지 신체적, 기술적 여건과 관계없이 웹 사이트를 통하여 원하는 서비스를 이용할 수 있도록 접근성을 보장하는 것으로서, 장애인, 고령자 등 정보 이용 접근이 어려운 사용자 뿐만 아니라, 일반 사용자들도 편리하게 웹 서비스를 이용할 수 있도록 차별없이 견고한 콘텐츠를 만드는 것을 말한다. 따라서 웹 접근성을 지키기 위한 가장 기본적이면서도 최선의 방법으로써 바로 웹 표준을 따라야 한다.

HTML5는 W3C (World Wide Web Consortium)이 2014년 10월 28일 확정된 HTML의 최신 표준이다. . HTML 4.01의 상위 버전으로써, 플래시나 실버라이트 등의 플러그인을 기반으로 하는 응용 프로그램에 대한 필요성을 줄이는 것에 초점을 맞추고 있다.

HTML5라고 불리우는 개념은 단순히 웹 문서를 작성할 때 사용되는 마크업 랭귀지(HTML)의 문법적 (syntactic) 버전 뿐만 아니라 새로운 DOM API 스펙을 포함한다. API 면에서 HTML5에서는 비디오 및 오디오와 같은 미디어 엘리먼트에 대한 API 뿐 아니라 오프라인 웹 앱 구현, 문서 편집 등과 같은 다양한 기능에 대한 API가 추가되었으며, WHATWG에 의해 Geolocation, Web SQL, File API, Audio API 등이 "Living standard"로 권고되고 있다. 이를 통해 이전에는 플래시, 실버라이트 등의 외부 플러그인을 통해서만 지원할 수 있던 클라이언트 사이드에서의 사용자 인터페이스를 위한 기능들의 상당수를 브라우저 자체의 기능을 이용해 구현할 수 있게 되었다.

## 2. DTD (Document Type Definition)

문서 유형을 정의하는 코드로써 문서의 첫 문장으로 사용된다

브라우저에게 문서의 유형(DOCTYPE)을 알려 올바르게 웹 페이지를 처리할 수 있도록 해 준다

문서의 유형을 알아야 문서의 유효성을 검사할 수 있고 개발자가 원하는 모습의 웹페이지를 제대로 표현할 수 있게 된다

가장 많이 사용되는 HTML 버전은 HTML 4.01과 XHTML 1.0, HTML 5 이며 각각의 DTD를 다르게 설정한다

### 2.1 HTML 4.01 DOCTYPE

- 이전 버전으로 제작된 HTML문서와의 호환성을 위한 DOCTYPE

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"  
"http://www.w3.org/TR/html4/loose.dtd">
```

- 정확한 표준 모드로 사용하기 위한 W3C의 권장 DOCTYPE

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"  
"http://www.w3.org/TR/html4/strict.dtd">
```

- 프레임 셋을 이용한 웹사이트를 만들 때 사용하는 DOCTYPE

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"  
"http://www.w3.org/TR/html4/frameset.dtd">
```

### 2.2 XHTML1.0 DOCTYPE

- 이전 버전으로 제작된 HTML문서와의 호환성을 위한 DOCTYPE

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

- 정확한 표준 모드로 사용하기 위한 W3C의 권장 DOCTYPE

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

- 프레임 셋을 이용한 웹사이트를 만들 때 사용하는 DOCTYPE

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
```

## 2.3 HTML5 DOCTYPE

- 이전 버전으로 제작된 HTML 문서와의 호환성을 위한 DOCTYPE

```
<!DOCTYPE HTML>
```

## 3. HTML 문서 구조

HTML은 태그로 이루어져 있으며 태그는 여는 태그와 닫는 태그로 이루어져 있다

닫는 태그가 없는 태그도 존재한다

태그는 소문자로 쓰고 태그의 속성명은 큰따옴표(" ")를 이용해 감싸주는 것을 권장한다

HTML 문서는 .html 확장자를 가진다

```
<여는태그 속성명="속성값">내용</닫는태그>
```

태그 안에 태그를 넣을 수 있지만 여는 태그와 닫는 태그는 순서에 맞게 한 쌍을 이루어야 한다

- 태그를 사용의 잘못된 예

```
<div><p>에러</div></p>
```

- 태그를 사용의 올바른 예

```
<div><p>정상</p></div>
```

## 3.1 기본 HTML

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

<html> 태그를 시작으로 해당 문서가 HTML 문서임을 정의한다

<head> 태그를 이용해 문서에 대한 정보들을 입력한다. 문서 제목, 문서 스타일, 문서 설정 등을 입력한다

<body> 화면에 나타나게 할 내용들을 입력한다

- 간단한 HTML 작성

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>제목</title>
</head>
<body>
  <h1>Hello</h1>
  <div>
    <p>Welcom to HTML</p>
    <p>함께 HTML을 배워봅시다</p>
    <!--주석 : 출력되지 않습니다 -->
  </div>
</body>
</html>
```

<!DOCTYPE html> : 문서의 버전이 HTML5 임을 브라우저에 알림

<html> : HTML 문서 전체를 감싸는 태그. <html> 태그 바깥에 다른 태그가 있으면 안 된다

<head> : HTML 문서에 대한 정보를 나타내는 부분. 하나만 존재해야 하고, <html> 태그 영역 바로 아래에 둔다

<meta charset="UTF-8"> : HTML 문서의 문자집합으로 "utf-8"을 사용함을 의미

<title> : 제목 표시줄에 나타날 내용 입력, 브라우저 상단 탭에 보여진다.

<body> : 브라우저에 실제로 보여지는 부분을 나타냄. 하나만 존재해야 한다

<h1> : body 안에서 단락의 제목을 표현하는 태그

<div> : 구역을 표시하는 태그. 브라우저 상 보여지진 않지만 <p>태그 영역을 하나의 구역으로 묶는다

<p> : 문단을 표시하는 태그

<!--comment --> : HTML 문서의 주석표현 방식이며 주석 처리된 내용은 브라우저가 해석하지 않는다



## 4. 영역 관련 태그

<div>와 <span> 태그를 이용하여 영역을 설정할 수 있다

웹 페이지의 레이아웃을 구성하고자 할 때 사용하는 태그

<div>는 줄 바꿈이 적용되어 이미 존재하는 태그의 다음 줄에 영역이 설정되며 <span> 태그는 줄 바꿈이 적용되지 않아 옆으로 영역이 붙는다

<div>는 사각형 박스로 구역을 설정하고 <span>태그는 문장 단위로 영역을 지정한다

### inline, block 요소

태그가 영역을 지정하는 방식에는 inline 방식과 block 방식이 있다

각각 Inline 요소와 Block 요소라고 부르며 인라인 요소는 문자의 일부분만을 선택해서 지정하며 블록 요소는 넓은 범위를 묶어서 지정한다

인라인 요소로는 <span>, <b>, <a>, <img> 등의 태그가 있다

블록 요소로는 <div>, <p>, <ol>, <ul>, <table> 등의 태그가 있다

inline과 block 속성의 설정은 style 속성을 이용하여 설정된다

- inline 요소 : style="display: inline;"
- block 요소 : style="display: block;"

## 5. 글자 입력 관련 태그

HTML 문서에서 글자 입력과 글자의 속성, 문단 설정, 정렬, 목록과 관련된 태그이다.

### 5.1 제목 태그 - <h숫자>

제목으로 사용될 문장을 표현하는 태그이다. 숫자는 1부터 6까지 제공된다.

<h1>, <h2>, <h3>, <h4>, <h5>, <h6>이 있다

<h1>이 가장 중요한 제목이 되며 <h6>으로 갈수록 덜 중요한 제목이 된다

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
</head>
<body>
<h1>제목1</h1>
<h2>제목2</h2>
<h3>제목3</h3>
<h4>제목4</h4>
<h5>제목5</h5>
<h6>제목6</h6>
</body>
</html>
```

**제목1**

**제목2**

**제목3**

**제목4**

**제목5**

**제목6**

## 5.2 문단 태그 - <p>, <br>

한 문단을 나타내는 <p> 태그는 Paragraph 의 줄임말이다.

줄바꿈을 표현하는 <br> 태그는 Break 의 줄임말이다.

태그 하나가 하나의 문단을 표현하므로 내용에 줄바꿈이 있어도 적용되지 않는다. 줄바꿈을 적용하기 위해서는 새로운 <p> 태그를 이용하거나 <br> 태그를 이용한다

<p> 태그를 이용하면 문단을 나누기 때문에 줄 간격이 생기며 <br>을 이용하여 줄바꿈을 하면 줄 간격이 생기지 않는다

- 내용상의 줄 바꿈은 출력시 적용되지 않으며 공백문자로 인식되어 띄어쓰기가 된다

```
<html>
<head> </head>
<body>
<p>
가나다라마바사
아자차카타파하
</p>
</body>
</html>
```

가나다라마바사 아자차카타파하

- 문단이 나누어지며 줄 간격이 생긴다

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
</head>
<body>
<p>가나다라마바사<p>아자차카타파하</p>
</body>
</html>
```

가나다라마바사  
아자차카타파하

- 문단이 나누어지지 않으며 줄바꿈이 발생하고 줄 간격이 없다

```
<html>
<head> </head>
<body>
<p>가나다라마바사<br>아자차카타파하</p>
</body>
</html>
```

가나다라마바사  
아자차카타파하

### 5.3 입력 내용을 그대로 보여주는 태그 - <pre>

<p> 태그와 유사하지만 HTML 문서 작성시 입력한 내용을 그대로 보여준다

줄 바꿈, 정렬 상태 등을 입력한 상태 그대로 보여주어 자유롭게 내용을 작성할 수 있는 장점이 있지만 글자의 색상이나 모양을 변경할 수 없는 단점이 있다

- 줄 바꿈이 적용되는 모습을 볼 수 있다

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
</head>
<body>
<pre>
가나다라마바사
아자차카타파하
</pre>
</body> </html>
```

## 5.4 글자 서식 태그

글자의 서식을 결정하는 태그이다.

여러 종류가 있으며 여러 태그를 중복해서 설정 가능하다

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
</head>
<body>
<p><i>기울임</i> <br>
    <b>굵게</b> <br>
    아래<sub>첨자</sub> <br>
    위<sup>첨자</sup> <br>
    <ins>밑줄</ins> <br>
    <strong>강조</strong> <br>
    <small>작게</small> <br>
</p>
</body>
</html>
```

이외에도 많은 서식태그가 있다

기울임  
굵게  
아래첨자  
위첨자  
밑줄  
강조  
작게

## 5.5 글자 속성 변경 태그 - <font>

입력한 글자의 색이나 크기 또는 글꼴을 변경해 주는 태그이다.

단, HTML5 에서는 font 태그를 지원하지 않는다. (CSS를 통해 글자의 속성을 변경한다)

### 글자 크기 설정 속성 - size

1(작음)부터 7(큼)까지 설정 가능하며 기본값은 3 이다.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
</head>
<body>
<font size="1">글자 크기는 1~7까지 설정 가능</font> <br>
<font size="5">글자 크기는 1~7까지 설정 가능</font> <br>
<font size="7">글자 크기는 1~7까지 설정 가능</font> <br>
</body>
</html>
```

글자 크기는 1~7까지 설정 가능

글자 크기는 1~7까지 설정 가능

글자 크기는 1~7까지 설정 가능

### 글꼴 설정 속성 - face

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
</head>
<body>
<font face="Dotum">설치되어 있는 폰트만 가능</font> <br>
<font face="Gulim">설치되어 있는 폰트만 가능</font>
</body>
</html>
```

설치되어 있는 폰트만 가능  
설치되어 있는 폰트만 가능

### 글자 색상 설정 속성 - color

color의 속성값으로 영문색상이름, RGB코드, HTML코드값이 올 수 있다

```
<!DOCTYPE html>  
<html>  
<head>  
<meta charset="UTF-8">  
</head>  
<body>  
<font color="red">빨강색</font> <br>  
<font color="blue">파랑색</font> <br>  
<font color="grey">회색</font>  
</body>  
</html>
```

빨강색  
파랑색  
회색

## 6. 목록 태그

목록을 표현하는 태그이다. <ol>, <ul>, <dl> 태그가 있다.

<ul> 태그는 unordered list 의 줄임말로써 순서가 없는 목록을 뜻하며 목록 항목 앞에 기호가 표시된다, <ol> 태그는 ordered list 의 줄임말로써 순서가 있는 목록을 의미하며, 목록 항목 앞에 숫자나 영문자, 로마문자로 순번을 표시한다. 목록의 항목은 <li>태그(list)를 이용하여 나타낸다 <dl> 태그는 define list 의 줄임말로써 어떤 용어와 용어에 대한 설명을 하나의 목록으로 묶어서 리스트를 표시할 때 사용한다. <dt> 태그는 용어, <dd> 태그는 용어에 대한 설명을 작성할 때 사용한다.

- <ul> : 점으로 구분되는 목록

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
</head>
<body>
<ul>
<li>하나</li>
<li>둘</li>
<li>셋</li>
</ul>
</body>
</html>
```

- 하나
- 둘
- 셋

- <ol> : 숫자로 구분되는 목록

```
<!DOCTYPE html>
<html>
<head>
```



```
<meta charset="UTF-8">
</head>
<body>
<ol>
<li>하나</li>
<li>둘</li>
<li>셋</li>
</ol>
</body>
</html>
```

1. 하나
2. 둘
3. 셋

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
</head>
<body>
<dl>
<dt>HTML5</dt>
<dd>HTML5는 HTML의 완전한 5번째 버전으로 월드 와이드 웹 의 핵심 마크업 언어이
다. </dd>
<dt>CSS3</dt>
<dd>CSS3는 기존의 CSS2가 갖지 못했던 화려하고 역동적인 면모를 추가하여 상자의 크기에
따른 말줄임 표시, 투명한 배경, 그림자 효과, 둥근 모서리, 그라디언트, 도형의 회전과 비틀기,
애니메이션 효과 등이 가능해진 것이다</dd>
</dl>
</body>
</html>
```

## 7. 하이퍼링크 태그 - <a>

클릭시 다른 페이지로 이동시키는 하이퍼링크를 적용시켜 주는 태그이다.

속성으로 href와 target이 있으며 href는 링크될 주소, target은 새 창을 띄울 위치를 지정한다

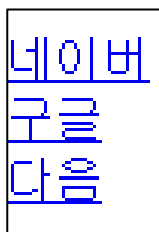
글자나 문단 뿐만 아니라 그림 또는 html 태그 요소 등에 적용할 수 있으며 적절한 범위를 고려하여 하이퍼링크를 적용해야 한다

### - <a>태그 기본 사용법

```
<a href="하이퍼링크 경로">하이퍼링크가 적용될 내용</a>
```

### - <a>태그 사용법

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
</head>
<body>
<a href="http://www.naver.com">네이버</a> <br>
<a href="http://www.google.com">구글</a> <br>
<a href="http://www.daum.net">다음</a>
</body>
</html>
```



### 7.1 링크열기 방식 target 지정

target 속성을 지정하면 하이퍼링크를 다양한 방법으로 열도록 할 수 있다

주로 기본값으로 두거나 \_blank만 사용된다

속성 값	설명
_blank	링크된 내용을 새로운 웹 브라우저 창에 열기
_self	링크된 내용을 현재 프레임에 열기(기본값)
_parent	링크된 내용을 현재 프레임을 호출한 상위 프레임에 열기(프레임 문서에 적용)
_top	링크된 내용을 프레임을 모두 없애고 웹 브라우저 화면 전체에 열기(프레임 문서에 적용)

## 7.2 <a> 태그를 이용한 책갈피 설정

문서 내에서 위치를 지정하고 그 위치로 이동할 수 있게 하는 기능을 책갈피라고 한다  
책갈피 이동은 <a href="#이동할요소의 ID">요소</a>를 이용한다

## 8. 테이블 태그 - <table>

표를 생성하는 태그로, 테이블의 셀 하나하나가 모두 태그로 이루어지며 복잡한 구조를 표현할 수도 있다

### 8.1 테이블 구조 태그

<table> 태그는 테이블을 명시하고 <thead>, <tbody>, <tfoot> 태그는 테이블의 영역을 나타낸다. <thead>, <tfoot> 태그는 테이블 태그 내에 한 개만 존재할 수 있다

<tr> 태그는 테이블의 한 행을 표현한다. <th>, <td> 태그는 <tr> 태그 안에 사용하며 <th> 태그는 제목에 해당하는 칸임을 표현하고, <td> 태그는 테이블의 일반적인 셀(칸)을 표현하는 태그이다. <caption>은 표의 제목을 나타내며 표에 대한 설명을 적는다

### 8.2 기본적인 테이블

<table>, <tr>, <td> 태그만을 이용하여 기본적인 테이블을 표현한다.

- 서울의 산을 테이블로 표현한 예제

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

```

<table>
<tr>
<td>이름</td>
<td>주소</td>
<td>전화번호</td>
</tr>
<tr>
<td>수락산</td>
<td>서울특별시 노원구 상계동</td>
<td>02-950-3900</td>
</tr>
<tr>
<td>도봉산</td>
<td>서울특별시 도봉구 도봉동</td>
<td>02-954-2566</td>
</tr>
<tr>
<td>불암산</td>
<td>서울특별시 노원구 상계동</td>
<td>02-950-3395</td>
</tr>
</table>
</body>
</html>

```

이름	주소	전화번호
수락산	서울특별시 노원구 상계동	02-950-3900
도봉산	서울특별시 도봉구 도봉동	02-954-2566
불암산	서울특별시 노원구 상계동	02-950-3395

### 8.3 테이블 행과 열 병합 – rowspan(행합치기), colspan(열합치기) 속성

테이블은 기본적으로 행별 열 갯수를 모두 동일하게 가져야 한다. 행별 열의 수가 맞지 않으면 제대로 된 테이블의 모습을 갖추지 못한다. 우선, 테이블이 생성되는 구조를 확인하기 위해 HTML 문서의 <head> 태그 안에 다음과 같이 적는다.

- 행별 열갯수가 맞지 않는 테이블 예제

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<style type="text/css">
table { border: 1px solid #ccc;}
th { border: 1px solid #ccc; }
tr { border: 1px solid #ccc; }
td { border: 1px solid #ccc; }
</style>
</head>
<body>
<table>
  <tr><td>1x1 셀</td> <td>1x2 셀</td></tr>
  <tr><td>2x1 셀</td> </td></tr>
</table>
</body>
</html>
```

- 브라우저에 나타나는 모습

1x1 셀	1x2 셀
2x1 셀	

두 번째 행의 셀이 한쪽에 치우쳐져 있는 것을 볼 수 있으며 2행 1열의 셀을 2행 전체에 걸쳐 표현하고 싶다면 해당 셀을 확장해야 한다. 셀을 확장할 때 사용하는 속성이 colspan과 rowspan이다. 행을 확장하려면 rowspan을 사용하며 열을 확장하려면 colspan을 사용한다

속성값은 확장하려는 셀의 크기만큼 지정해주면 된다

- 두 번째 행을 확장한 테이블 예제

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<style type="text/css">
table { border: 1px solid #ccc;}
th { border: 1px solid #ccc; }
tr { border: 1px solid #ccc; }
td { border: 1px solid #ccc; }
</style>
</head>
<body>
<table>
  <tr>
    <td>1x1 셀</td>
    <td>1x2 셀</td>
    <td>1x3 셀</td>
  </tr>
  <tr>
    <td colspan="3">2x1 셀</td>
  </tr>
</table>
</body>
</html>
```

- 브라우저에 나타나는 모습

1x1 셀	1x2 셀	1x3 셀
2x1 셀		

## 8.4 복잡한 구조의 테이블

- 서울의 산을 테이블로 표현한 예제

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<style type="text/css">
table { border: 1px solid #ccc;}
th { border: 1px solid #ccc; }
tr { border: 1px solid #ccc; }
td { border: 1px solid #ccc; }
</style>
</head>
<body>
<table>
  <caption>한국의 산</caption>
  <thead>
    <tr>
      <th>이름</th>
      <th>주소</th>
      <th>전화번호</th>
    </tr>
  </thead>
  <tfoot>
    <tr>
      <td colspan="3"><small>등산과 함께 몸도 마음도 건강하게!</small></td>
    </tr>
  </tfoot>
  <tbody>
    <tr>
      <th scope="row">수락산</th>
      <td>서울특별시 노원구 상계동</td>
      <td>02-950-3900</td>
```

```

</tr>
<tr>
  <th scope="row">도봉산</th>
  <td>서울특별시 도봉구 도봉동</td>
  <td>02-954-2566</td>
</tr>
<tr>
  <th scope="row">불암산</th>
  <td>서울특별시 노원구 상계동</td>
  <td>02-950-3395</td>
</tr>
</tbody>
</table>
</body>
</html>

```

<thead>, <tbody>, <tfoot> 을 이용해 테이블의 영역을 지정  
 테이블의 제목에 해당하는 영역에 <th> 적용  
 산 이름에 해당하는 셀에 행의 제목을 나타내는 속성인 scope="row"를  
 <tfoot> 영역을 colspan="3" 속성을 적용하여 3개의 <td>셀을 합친 효과 적용

#### - 테이블 스타일 적용

```

<style type="text/css">
table { border: 1px solid #ccc;}
th { border: 1px solid #ccc; }
tr { border: 1px solid #ccc; }
td { border: 1px solid #ccc; }
</style>

```

테이블의 테두리를 지정하기 위해 CSS를 이용한 스타일을 적용하였다  
 <head> 태그 내에 입력  
 테이블 전체의 테두리와 행, 열, 제목 셀 모두 회색(#ccc) 1px 두께의 실선으로 테두리 설정

#### - 브라우저에 나타나는 모습



## 한국의 산

이름	주소	전화번호
수락산	서울특별시 노원구 상계동	02-950-3900
도봉산	서울특별시 도봉구 도봉동	02-954-2566
불암산	서울특별시 노원구 상계동	02-950-3395
등산과 함께 몸도 마음도 건강하게!		

## 9. 폼 태그 - <form>

회원가입이나 로그인처럼 정보를 서버로 보내는 역할을 가진 양식 태그이다.

<form> 태그로 정보를 보낼 부분을 감싸주어야 한다. action, method 두 가지 속성을 주로 사용한다. action속성은 <form> 태그의 정보를 보낼 서버의 주소를 적고, method 속성은 정보를 보낼 방식을 적는 부분이다. method는 주로 post와 get을 사용한다.

\* get METHOD : URI를 통해 정보를 전달

\* post METHOD : HTTP Request Body를 통해 전달

- <form> 태그 기본 구조

```
<form action="서버주소" method="메소드형식"> 보낼 정보 </form>
```

보낼 정보는 <input>, <select>, <option>, <textarea> 등의 태그를 이용하여 입력받는다

- 인적사항을 입력 받는 <form>태그

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
</head>
<body>
  <form action="#" name="info" method="get">
    <fieldset style = "width:150">
      <legend>개인 정보</legend>
      이름 : <input type = "text" name = "name"/><br><br>
```

```

나이 : <input type = "text" name = "age"/> <br> <br>
</fieldset>
<br>
<fieldset style = "width:180; height:180">
<legend>특이 사항</legend>
취미 : <input type = "text" name = "hobby"/> <br> <br>
특기 : <input type = "text" name = "specialty"/>
</fieldset>
</form>
</body>
</html>

```

## 9.1 <label> 태그

폼 태그 안에서 정보를 입력 받는 태그가 어떤 역할을 하는지 알려주는 이름표 태그

서버로 정보를 전달하는 역할은 하지 않으며 화면 상 어떤 정보를 입력해야 하는 지 알리기 위한 목적으로 사용한다

<label> 태그의 for 속성을 이용하여 <input> 태그의 id와 같게 속성값을 주면 한 쌍으로 묶이게 된다

## 9.2 <input> 태그

폼 안에서 정보를 입력받을 부분을 지정하는 입력양식 태그이다.

<input> 태그의 속성으로 type, placeholder, name, value, id 등을 주로 사용한다

type 속성은 정보를 어떤 형식으로 받을지를 정하며 대표적으로 text, radio, checkbox를 이용한다. text 속성을 세분화하여 text, email, date, time, datetime, password, number, range, search, url, week, month 등을 사용한다. <input>태그에서 받을 정보의 종류에 따라 다른 type 속성값을 사용한다.

placeholder 속성은 <input> 태그에 아무런 값이 입력되지 않았을 때 나타나는 글을 정하는 속성으로 도움말 기능을 할 수 있다

name 속성은 <input>태그를 통해 정보를 서버에 전달하면 정보에 대한 키로써 같이 전달되는 값을 지정한다.

value 속성은 전달되는 값을 나타낸다

id 속성은 <input> 태그를 다른 태그와 구분하기 위한 목적으로 사용된다

### text type

기본적인 문장 정보를 입력받을 때 사용

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
</head>
<body>
<input type="text" placeholder="아이디 입력" id="userId" name="userId">
</body>
</html>
```

### password type

비밀번호 정보를 입력받을 때 사용

input 창에 입력을 하면 입력한 값이 화면에서 보이지 않게 된다

```

<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
</head>
<body>
<input type="password" placeholder="비밀번호 입력" id="userPwd" name="userPwd">
</body>
</html>

```



### checkbox type

항목을 체크할 수 있도록 생성

<label> 태그의 for 속성을 checkbox <input> 태그의 id와 같은 값으로 설정하면 label영역을 클릭해도 체크박스가 체크된다

```

<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
</head>
<body>
<label for="chkBox">체크</label>
<input type="checkbox" id="chkBox" name="chkBox">
</body>
</html>

```

체크 ☐

for 속성으로 묶지 않고 <label> 태그 안에 <input>태그를 넣어도 한 쌍이 된다

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
</head>
<body>
<label>체크
<input type="checkbox" id="chkBox" name="chkBox">
</label>
</body>
</html>
```

### radio type

여러 개의 선택사항 중 한가지를 선택할 수 있는 라디오버튼 생성

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
</head>
<body>
<label>성별</label>
<input type="radio" name="gender" value="m">남
<input type="radio" name="gender" value="f">여
</body>
</html>
```

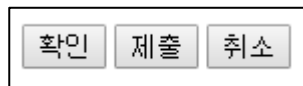
성별 ☐ 남 ☐ 여

## button type

버튼을 생성한다

버튼을 생성하는 type의 속성값으로 button, submit, reset이 있다

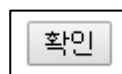
```
<html>
<head> </head>
<body>
<input type="button" id="btnOk" value="확인">
<input type="submit" id="btnSubmit" value="제출">
<input type="reset" id="btnReset" value="취소">
</body>
</html>
```



submit과 reset 속성값은 특수한 기능을 가지는데 submit 버튼은 <form> 태그의 action 속성값인 서버주소로 <form> 태그 내의 정보를 전달하는 기능을 한다. reset 속성의 버튼은 <form>태그 내에 입력된 정보들을 초기화시키는 기능을 가지고 있다.

<input> 태그를 이용하지 않고 <button>태그를 이용하여 버튼을 생성할 수도 있다. 하지만 <button> 태그를 이용하여 폼 양식을 전송하려면 추가적인 자바스크립트가 필요하다.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
</head>
<body>
<button id="btnOk">확인</button>
</body>
</html>
```

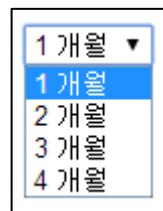
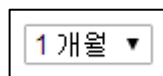


### 9.3 <select> - <option> 태그

여러 옵션 중 선택할 수 있도록 리스트박스를 생성하는 태그

선택된 옵션값을 전달할 때 같이 전송할 키 값이 되는 name 속성은 <select>태그에서 설정하며 전달될 값인 value 속성은 <option>태그에서 설정한다

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
</head>
<body>
<select name="month">
<option value="one">1 개월</option>
<option value="two">2 개월</option>
<option value="three">3 개월</option>
<option value="four">4 개월</option>
</select>
</body>
</html>
```



## 9.4 <textarea> 태그

텍스트 입력상자

여러 줄로 된 텍스트를 입력받을 때 사용하며 공백과 개행을 입력한 그대로 받아들인다  
readonly, cols, rows, placeholder 등의 속성을 사용한다

- 50개의 문자를 입력할 수 있는 너비와 3줄의 형태를 가지는 <textarea> 영역 생성

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
</head>
<body>
<textarea rows="3" cols="50" placeholder="긴글 입력" name="content"> </textarea>
</body>
</html>
```

## 10. 이미지 태그 - <img>

이미지를 넣고자 할 때 사용하는 태그

```
<img src="" width="" height="" border="" vspace="" hspace="" align="" alt="" title="">
```

src : 보여줄 이미지의 주소. 필수 속성

width : 이미지의 너비. 픽셀(px) 또는 %로 값 부여

height : 이미지의 높이. 픽셀(px) 또는 %로 값 부여

border : 이미지 테두리의 굵기 (HTML5에서 지원하지 않는다)

vspace : 이미지와 텍스트 사이의 위아래 여백 (HTML5에서 지원하지 않는다)

hspace : 이미지와 텍스트 사이의 좌우 여백 (HTML5에서 지원하지 않는다)

align : 텍스트와의 상관관계 위치. top, middle, bottom, left, right가능(HTML5에서 지원하지 않는다)

alt : 이미지가 없거나 로딩되지 않을 때 대체되는 텍스트

title : 마우스를 이미지에 올려 놓았을 때 팝업 도움말로 보이는 텍스트



## 11. 비디오 태그 - <video>

웹 페이지에서 동영상을 볼 수 있게 만들어 주는 기능을 수행

HTML5 이전에는 플러그인을 사용해야만 가능했지만 HTML5에서는 비디오 태그를 통해 동영상을 보여줄 수 있다

<video> 태그의 src 속성을 이용하여 비디오를 추가하거나 src 속성을 설정하지 않고 <source> 태그를 두어 비디오를 추가 할 수 있다

```
<video src="" poster="" width="" height="" controls="controls" autoplay="autoplay" loop="loop">
</video>
```

src : 비디오 파일의 경로 지정

poster : 비디오가 준비중일 때 보여질 이미지 파일의 경로

width : 비디오의 너비

height : 비디오의 높이

controls : 비디오 재생 도구를 출력할 지 결정

autoplay : 비디오를 자동 재생할 지 결정

loop : 비디오를 반복 재생할 지 결정

<video>태그의 src속성을 설정하지 않고 <source>태그를 이용하여 비디오를 추가할 수도 있다. <source>태그를 이용하여 비디오를 추가할 경우 브라우저의 코덱 상황에 맞게 비디오를 선택하여 플레이하므로 여러 방식으로 인코딩된 동영상을 넣어주어 사용한다.

- <source> 태그를 이용한 방법

```
<video poster="" width="" height="" controls="controls" autoplay="autoplay" loop="loop">
  <source src="" type="">
  <source src="" type="">
</video>
```

type : 비디오 파일의 코덱 정보

## 12. 오디오 태그 - <audio>

웹 페이지에서 소리 콘텐츠를 재생하기 위한 태그

HTML5부터 사용할 수 있는 태그

<video> 태그와 마찬가지로 <source> 태그를 이용할 수 있다

```
<audio src="" controls="controls" autoplay="autoplay" loop="loop"> </audio>
```

src : 오디오 파일의 경로 지정

controls : 오디오 재생 도구를 출력할 지 결정

autoplay : 오디오를 자동 재생할 지 결정

loop : 오디오를 반복 재생할 지 결정

## 13. 프레임 태그 - <iframe>

웹 페이지에 콘텐츠 삽입을 목적으로 사용되는 태그

문서 내에 다른 문서를 포함시킬 수 있다

```
<iframe src="" srcdoc="" width="" height="" sandbox="">대체내용</iframe>
```

src : 프레임의 내에 보여질 경로 지정. 유효한 URL이어야 한다

width : 프레임의 너비

height : 프레임의 높이

sandbox : 추가되는 프레임에 제한을 두는 속성. 보안성을 높이기 위해 사용되는 HTML5에서 추가된 속성

srcdoc : 프레임의 내에 보여질 내용 지정. 올바른 HTML이나 XML형식을 가져야한다. HTML5부터 사용 가능

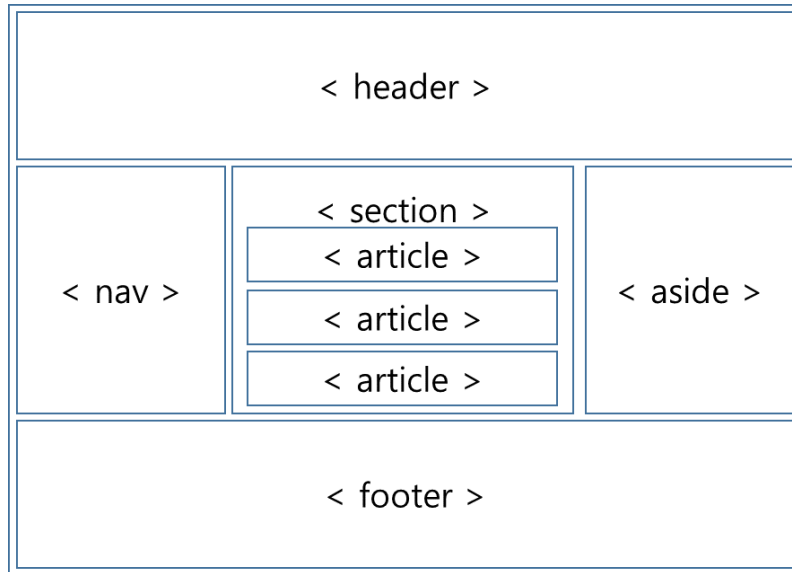
## 14. HTML5 - 시맨틱 태그

문서 구조를 정의하는 HTML5 에서 새로 추가된 태그이다.

<body> 태그 내에서 영역을 구분하는 용도로 사용된다

기본적으로 <div> 태그와 같은 역할을 하지만 시맨틱 태그가 가지는 의미는 검색엔진이나 그 이외의 기계적인 동작들을 수행될 때 웹 페이지를 쉽게 이해할 수 있게 해준다

### - 주요 HTML5 시맨틱 태그



<header> : 헤더 부분. 사이트 소개, 로고, 메뉴, 머리글 등

<footer> : 푸터 부분. 사이트 제작자나 저작권 관련 정보 등

<nav> : 네비게이터 부분. 사이트 내 메뉴 등

<section> : 실제 문서 내용

<article> : 문서 내용이 많을 경우 <section> 영역을 여러 개의 주제별로 <article>로 나눔

<aside> : 사이드 바와 같은 형태의 영역

## **Chapter 2. CSS3을 활용한 웹 디자인 및 스타일 꾸미기**

# 1. CSS란

Cascading Style Sheets의 약자로 마크업 언어가 실제 표시되는 방법을 기술한 언어이다.

HTML, XHTML, XML 같은 마크업 언어 문서의 스타일을 꾸밀 때 사용하는 스타일 시트이다.

## 1.1 CSS2 속성

### 일반

속성	설명	세부옵션
display	원소 프레임 유형	block   inline   list-item   none
overflow	원소 내용이 넘칠 때 설정	visible   hidden   scroll   auto
visibility	원소 보임 설정	visible   hidden

### Background

속성	설명	세부옵션
background-color	배경색깔	절대값/상대값 표현
background-image	배경이미지	url("")
background-repeat	배경이미지 반복여부	repeat no-repeat repeat-x repeat-y
background-attachment	배경이미지 스크롤 여부	scroll fixed
background-position	배경이미지 시작위치	(x y), top, center,bottom,left,right

### Border

속성	설명	세부옵션
border-width	border 너비	절대값/상대값 표현
border-style	border 종류	dashed dotted double groove inset outset redge solid none
border-color	border 색깔	색상명/RGB/HEXA 표현

### Text

속성	설명	세부옵션
letter-spacing	자간 간격	절대값/상대값 표현
line-height	행간격	절대값/상대값 표현
text-align	수평정렬여부	left right center justify
text-decoration		blank line-through none overline underline

text-indent	들여쓰기여부	절대값/상대값 표현
text-transform	알파벳대소문자처리	capitalize lowercase uppercase
vertical-align	수직정렬	baseline length sub super top text-top middle bottom text-bottom px
word-spacing	단어간 간격	절대값/상대값 표현

## Font

속성	설명	세부옵션
font-style		italic normal
font-variant	small-cap설정 시 알파벳소문 자를 크기작은 대문자로 변경,	normal small-cap
font-weight	글꼴 굵기	bold normal lighter bolder integer[100-900]
font-size	글꼴 크기	절대값/상대값 표현
font-family	글꼴 종류	절대값/상대값 표현

## Margin

속성	설명
margin	요소간 바깥거리
margin-top	요소간 위쪽 바깥거리
margin-right	요소간 오른쪽 바깥거리
margin-bottom	요소간 아래쪽 바깥거리
margin-left	요소간 왼쪽 바깥거리

## Padding

속성	설명
padding	요소 안 여백설정
padding-top	요소 안 위쪽 여백
padding-right	요소 안 오른쪽 여백
padding-bottom	요소 안 아래쪽 여백
padding-left	요소 안 왼쪽 여백

## List

속성	설명	세부옵션
list-style-type	목록 항목 형식표시	circle   lower-alpha   square   lower-alpha
list-style-position	목록 배치 위치 설정	inside   outside
list-style-image	목록 태그 이미지 설정	url("")

## Dimension

속성	설명
height	요소 높이
width	요소 너비
max-height	요소 최대높이
max-width	요소 최대너비
min-height	요소 최소높이
min-width	요소 최소너비

## Position

속성	설명	세부옵션
clear	float속성에 대한 불허처리	none   left   right   both
float	요소 정렬 기준 설정	none   left   right
left	요소 바깥거리 밖으로 왼쪽거리	절대값/상대값 표현
top	요소 바깥거리 밖으로 위쪽거리	절대값/상대값 표현
position	요소 포지셔닝 설정	static   relative   absolute
z-index	요소들간 덮어쓰기 높이값	auto, integer[higher number on top]

## Table

속성	설명	세부옵션
border-collapse	경계선 합병 여부	separate   collapse
border-spacing	인접 셀 경계 사이거리	절대값/상대값 표현
caption-side	캡션 위치	top   bottom
empty-cells	빈 셀로 테두리/배경설정	show   hide
table-layout	표 레이아웃	auto   fixed

## 2. CSS3

Cascading Style Sheets(CSS) 언어의 가장 최신 버전이고 CSS2.1을 확장하며, 모듈기반으로 개발 중인 표준이다. 다중열(multi-columns), 유동적인 상자(flexible box), 격자 배치(grid layouts) 뿐만 아니라 둥그런 모서리(rounded corners), 그림자(shadows), 부드러운 색의 변이(gradients), 변이(transitions), 움직임(animations) 등을 새로 지원한다. 실험적인 부분은 브라우저 공급자의 구현에 따라 다를 수 있다

### 2.1 CSS3에 추가된 속성

#### Multi-columns

속성	설명	세부옵션
column-count	컬럼수 지정	number auto
column-gap	컬럼간 간격	절대값/상대값 표현
column-rule-style	컬럼간 구분선 스타일	none hidden dotted dashed solid double groove ridge inset outset
column-rule-width	컬럼간 구분선 너비	절대값/상대값 표현
column-rule-color	컬럼간 구분선 색깔	색상명/RGB/HEXA 표현
column-rule	컬럼간 구분선 속성 모음	column-rule-width column-rule-style column-rule-color
column-span	표현될 컬럼개수	none   all
column-width	컬럼너비	절대값/상대값 표현

#### Flexible box

속성	설명	세부옵션
flex-direction	아이템이 전개되는 순서	row   row-reverse   column   column-reverse
flex-wrap	아이템이 한줄, 또는 여러줄에 나열될지 여부	nowrap   wrap   wrap-reverse
flex-flow	direction과 wrap을 모아서 표현	flex-direction flex-wrap



justify-content	아이템 수평 정렬방식	center   flex-start   flex-end   space-around
align-items	아이템 수직 정렬방식	baseline   center   flex-start   flex-end   stretch
align-content	아이템 행간 정렬방식	stretch   center   flex-start   flex-end   space-around
order	아이템 순서	number
flex-grow	아이템너비	number
flex-shrink	아이템너비 상대값	number
flex-basis	아이템 너비 절대값	number
align-self	아이템 기본정렬 재정의	auto   baseline   center   flex-start   flex-end   stretch

## Grid layouts

속성	설명	세부옵션
grid-column	열 설정, start, end 동시표현	grid-column-start / grid-column-end
grid-row	행 설정, start, end 동시표현	grid-row-start / grid-row-end
grid-area	행의 시작, 끝, 열의 시작, 끝값 표현. grid-row-start, grid-row-end, grid-column-start, grid-column-end 동시표현	
grid-row column-gap	그리드간 간격설정 grid-row-gap, grid-column-gap 동시표현	

## Rounded corners

속성	설명	세부옵션
border-radius	네 모서리값 동시표현.	절대값/상대값 표현
border-top-left-radius	좌측상단 모서리 굴림처리	절대값/상대값 표현
border-top-right-radius	우측상단 모서리 굴림처리	절대값/상대값 표현
border-bottom-right-radius	우측하단 모서리 굴림처리	절대값/상대값 표현
border-bottom-left-radius	좌측하단 모서리 굴림처리	절대값/상대값 표현

## Shadows

속성	설명	세부옵션
text-shadow	글자관련 그림자처리.	수평그림자값 수직그림자값 번짐값 컬러값
box-shadow	요소관련 그림자처리	수평그림자값 수직그림자값 번짐값 컬러값

## Gradients

속성	설명	세부옵션
linear-gradient	선형 그라디언트	방향 컬러값...
radial-gradient	원형(타원형) 그라디언트	원/타원여부 컬러값...

## Transitions

속성	설명	세부옵션
transition-property	트랜지션이 일어날 css속성	css속성...
transition-duration	트랜지션 진행시간	초단위시간값
transition-timing-function	트랜지션간 진행속도 옵션	linear ease ease-in ease-out ease-in-out cubic-bezier(n,n,n,n)
transition-delay	트랜지션이 일어나기까지 지연시간	초단위 시간값

## Animations

속성	설명	세부옵션
@keyframes	애니메이션코드	animationname {keyframes-selector {css-style}}
animation-delay	시작전 대기시간	time
animation-direction	애니메이션 방향성	normal reverse alternate alternate-reverse
animation-duration	애니메이션 진행시간	time
animation-fill-mode	정지모드에서 요소상태 옵션	none forwards backwards both
animation-	반복회수	number infinite

iteration-count		
animation-play-state	애니메이션 진행여부	paused running
animation-timing-function	애니메이션 진행속도 옵션	linear ease ease-in ease-out ease-in-out step-start step-end steps(int,start end) cubic-bezier(n,n,n,n)

### 3. CSS를 HTML문서에 적용하는 방법

#### 3.1 인라인 방식

태그의 속성으로 style을 지정해 직접 스타일을 적용하는 방법

태그에 직접 작성하기 때문에 간편하게 작성할 수 있고 바로 확인할 수 있다는 장점이 있지만 개별적으로 적용되어있어 유지보수가 어려워 권장되지 않는 방식이다

```
<p style="color: red;"> 내용 </p>
```

#### 3.2 임베디드 방식

<head>태그 내부에 <style>태그를 생성해 문서의 태그 및 클래스 등에 스타일을 지정하는 방법

```
<head>
<style type="text/css">
  p {
    color: red;
  }
</style>
</head>
```

#### 3.3 css 파일을 로드해 적용하는 방식

별도로 CSS파일을 작성해 HTML문서에 로드하여 스타일을 적용하는 방법이다. 스타일시트에 대한 관리가 편리해진다. 외부 파일의 확장자는 .css로 생성한다

@import를 이용하는 방식과 <link>태그를 이용하는 방식이 있으며 @import방식보다는 <link>태그를 이용한 방식을 권장한다. @import방식보다 <link>태그가 css파일을 불러올 때 속도적인 측면에서 유리하며 HTML문서에 로드된 스타일시트가 직관적으로 보이기 때문에 유지보수가 용

이해진다

### **@import 방식**

<style> 내부에서 사용할 수 있으며 개별 CSS파일을 서로 내부에서 불러올 수 있다는 장점이 있다

- board.html

```
<style type="text/css">
@import url(board.css)
</style>
```

- board.css

```
@import url(main.css)
.board {
  background-color: #FF0;
}
```

- main.css

```
body {
  color: #333;
  padding: 0;
  margin: 10px;
}
```

### **link 방식**

HTML문서의 <head>태그에 <link> 태그를 추가하는 방식

@import방식은 CSS파일 내부에서도 임포트할 수 있지만 개별적인 파일로 따로 불러와야 한다

```
<head>
<link href="main.css" rel="stylesheet" type="text/css">
<link href="board.css" rel="stylesheet" type="text/css">
</head>
```

## 4. 색상과 단위

### 4.1 색상 단위

CSS 색상값을 입력할 때 사용하는 색상 단위로 5가지가 있다

단위	설명	예
색상 이름	색상의 이름을 이용	red, blue 등
rgb(red, green, blue)	red, green, blue 값 이용	rgb(255, 0, 0), #ff0000
rgba(red, green, blue, alpha)	rgb에 alpha(투명도) 추가	rgba(255, 0, 0, 0.3)
hsl(hue, saturation, lightness)	색조, 채도, 명도를 이용	hsl(120, 100%, 50%)
hsla(hue, saturation, lightness, alpha)	hsl에 alpha(투명도) 추가	hsla(120, 100%, 25%, 0.3)

#### 색이름 방식

색을 표현할 때 색상 이름을 사용하는 방식

black, white, red, green, tomato 등이 있다

#### RGB 방식

red, green, blue 색상 값을 조합하여 색을 표현하는 방식

#을 붙여 16진수 3자리나, 6자리로 표현하거나 0~255의 값 또는 0~100%의 값을 이용하여 RGB(0, 0, 0)형식으로 표현할 수 있다

#### RGBA 방식

RGB방식에 투명도가 포함된 방식

alpha(투명도) 값은 0.0(완전투명)에서 1.0(완전 불투명)의 값을 사용한다

#### HSL 방식

hue(색조), saturation(채도), lightness(명도)를 조합하여 색을 표현하는 방식

색조 값은 0~360사이의 숫자이며 0 또는 360은 red, 120은 green, 240은 blue이다

채도 값은 0(회색 빛)~100%(원래의 완전한 색상)로 지정한다

명도 값은 0(검정색, 어두운)~100%(흰색, 밝은)로 지정한다

#### HSLA 방식

HSL방식에 투명도가 포함된 방식

alpha(투명도) 값은 0.0(완전투명)에서 1.0(완전 불투명)의 값을 사용한다

## 4.2 웹 안전 색상 (Web Safty Color)

표현 가능한 색상은 16,777,216가지(rgb기준)가 된다. 운영체제나 브라우저의 종류에 따라 색을 제대로 표현되지 않을 때가 있는데 이러한 색의 왜곡현상을 줄이기 위해 대폭 줄여서 표준으로 웹 안전색상을 정했다.

RGB 색상 요소는 red, green, blue 색상이 각각 256가지씩 존재하는데 이를 6단계로 나누어 총 216색상으로 줄여서 표준으로 정했다. 0~255값을 16진수 '00, 33, 66, 99, CC, FF'로 표현하게 된다. 개발자나 디자이너가 정한 색상이 표현되지 않을 경우가 적어 안전(safe)하지만 표현할 수 있는 색상이 제한적이게 된다

## 4.3 크기 단위

### 고정 크기 단위

부모 요소나 기타 요소들에 영향을 받지 않고 일정한 크기를 유지하는 단위

- in : 인치를 뜻한다. 현실세계의 인치와는 다르다. 사용자나 운영체제의 설정에 따른 논리적인 크기
- px : 화소 단위. 정확하게 배치하거나 크기를 잡을 때 사용하기 좋다
- pt : 포인트. 1pt = 1/72in
- pc : 파िका. 1pc = 1/6in
- cm : 논리 센티미터. 1in = 2.54cm
- mm : 논리 밀리미터. 1in = 25.4mm

### 가변 크기 단위

상대적인 크기를 가지는 단위

- em : 요소에 지정하는 글자 크기 단위. 부모 요소에 지정한 글자 크기를 기준으로 배율을 조정한다. 2em일 경우 부모 요소의 글자크기의 2배. 글자 크기에 따라 레이아웃을 유동적으로 만들 때 사용한다
- ex : 요소에 들어있는 폰트의 소문자 'x'의 높이를 기준으로한 비율. em과 연관이 있으며 1em은 1ex보다 약 두배 크다. 거의 쓰이지 않는다
- rem : root em이라는 뜻으로, HTML문서의 root 요소인 <html> 태그를 나타내며 루트 태그에 지정된 크기를 기준으로 상대적인 값을 가지게 된다
- % : 요소의 크기의 비율을 따져 크기를 가진다

## 5. 선택자 (Selector)

선택자는 특정 요소를 선택하여 스타일을 적용할 수 있게 해준다

CSS에서 스타일을 정의할 때 선택자를 이용하여 요소를 선택하고 요소를 어떻게 표현할 것인지 선언하여 브라우저에게 알려준다. 선택자와 스타일 선언문을 묶어 CSS Rule-set 이라고 부르며 CSS Rule-set이 모여 Style Sheet가 된다

- CSS Rule-set 예시

```
<style>
p { color: red; padding: 5px; }
</style>
```

p : <p> 태그 선택자

{ } : CSS 선언 블록

color, padding : 속성

red, 5px : 속성 값

### 5.1 선택자 종류

전체 선택자

선택자 패턴	의미
*	HTML페이지 내부의 모든 태그를 선택

HTML 페이지 내부의 모든 요소(태그)에 같은 CSS속성을 적용할 때 사용한다. 주로 margin이나 padding같은 값을 초기화하거나 기본값을 정해줄 때 사용한다. 문서 내의 모든 요소에 적용해야 하므로 페이지 로딩이 느려질 수 있어 자주 사용하지 않는 것이 좋다.

- 전체 선택자 예제

```
<style>
/* CSS */
* { margin: 0; padding 0; }
</style>
```

태그 선택자

선택자 패턴	의미
E	태그명이 E인 특정 태그를 선택

HTML 요소를 직접 지칭하는 간단한 선택자  
HTML 문서에서 사용한 태그를 모두 선택한다

- 전체 선택자 예제

```
<style>
/* CSS */
p { background: yellow; color: green; }

<!--HTML -->
<p>태그 선택자</p>
<div>
  <p>적용 영역</p>
  <pre>미 적용 영역</pre>
</div >
</style>
```

**클래스 선택자**

선택자 패턴	의미
.myClass	클래스 속성값이 myClass로 지정된 요소를 선택

특정 class 속성값을 가지는 요소를 선택한다. 클래스이름 앞에 .(마침표)를 추가해서 작성한다. 해당 클래스명으로 속성을 가지는 모든 요소를 선택한다. 클래스 선택자 앞에 태그명을 붙여주면 해당 태그 요소의 범위 내에서 해당 클래스를 선택한다.

- 클래스 선택자 예제

```
<style>
/* CSS */
.class1 { background: yellow; color: green; }
div.class2 { background: red; color: blue; }

<!--HTML -->
<p class="class1">클래스 선택자 </p>
<p class="class2">클래스 선택자 미 적용</p>
<div class="class2">클래스 선택자 적용</div >
</style>
```



## 아이디 선택자

선택자 패턴	의미
#myId	아이디 속성값이 myId로 지정된 요소를 선택

특정 id 속성값을 가지는 요소를 선택한다. #을 사용하여 id 속성값을 찾아 선택한다

클래스 선택자는 여러 번 반복될 필요가 있는 스타일일 경우 사용하고, 아이디 선택자는 유일하게 적용되는 스타일일 경우 사용한다

아이디 선택자의 우선순위가 클래스 선택자보다 높으므로 우선 적용되어야 할 스타일을 아이디 선택자로 정의하는 것이 좋다

### - 아이디 선택자 예제

<pre> &lt;html&gt; &lt;head&gt; &lt;style&gt; /* CSS */ #id1 { background: yellow; color: green; } div#id2 { background: red; color: blue; } &lt;/style&gt; &lt;/head&gt; &lt;body&gt; &lt;p id="id1"&gt;아이디 선택자&lt;/p&gt; &lt;p id="id2"&gt;아이디 선택자 미 적용&lt;/p&gt; &lt;div id="id2"&gt;아이디 선택자 적용&lt;/div &gt; &lt;/body&gt; &lt;/html&gt; </pre>
--

## 복합 선택자

두 개 이상의 요소가 모인 선택자

태그들의 관계를 따져 요소를 선택한다

### - 하위 선택자와 자식 선택자

선택자 패턴	의미
E F	E 요소의 하위 요소 F를 선택
E > F	E 요소의 자식 요소 F를 선택

계층적으로 구성되어있는 HTML 요소들의 관계에서 상위 계층의 요소를 부모 요소라고 하며 하위 계층의 요소를 자식 요소라고 한다

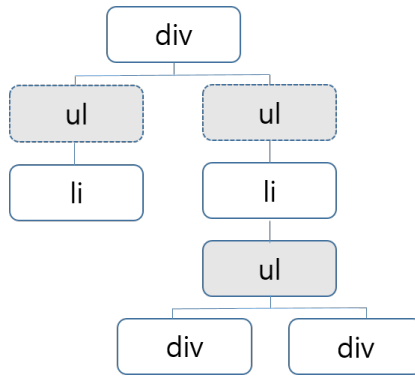
요소 두 개를 나란히 두어 사용하는 하위 선택자는 부모 요소의 모든 하위 요소를 선택한다

요소 사이에 > 기호를 사용하는 자식 선택자는 부모 요소의 바로 아래 자식 요소만 선택한다

#### - 하위, 자식 선택자 예제

```
<html>
<head>
<style>
/* CSS */
div ul { background: yellow; }
div > ul { border: 1px dotted red; }
</style>
</head>
<body>
<!--HTML →
<div>
  <ul> <li>1 </li> </ul>
  <ul>
    <li>2
      <ul>
        <li>2-1 </li>
        <li>2-2 </li>
      </ul>
    </li>
  </ul>
</div>
</body>
</html>
```

div요소의 하위 모든 ul 요소의 배경색이 노란색이 적용되고 div요소의 바로 아래 자식만 테두리를 가진다



- 인접 형제 선택자와 일반 형제 선택자

선택자 패턴	의미
E+F	E 요소를 곧바로 뒤따르는 F요소 선택 (E요소와 F요소 사이에 다른 요소가 존재하면 선택하지 않음)
E~F	E 요소 뒤따르는 모든 F요소 선택

같은 부모를 가지는 요소를 형제 관계라고 한다

- 형제 선택자 예제

```

<html>
<head>
<style>
/* CSS */
h1+p { background: green; color: yellow; }
h1~p { border: 1px solid red; }
</style>
</head>
<body>
<!--HTML -->
<p>선택되지 않음</p>
<h1>형제 선택자</h1>
<p>h1의 인접 형제</p>
<p>h1의 일반 형제</p>
<p>h1의 일반 형제</p>
</body>
</html>
  
```

## 속성 선택자

선택자 패턴	의미
E[attr]	"attr"속성이 포함된 요소 E를 선택
E[attr="val"]	"attr"속성 값이 정확하게 "val"과 일치하는 요소 선택
E[attr~="val"]	"attr"속성 값이 "val" 단어를 포함하는 요소 선택 (앞이나 뒤에 공백이나 하이픈('-')으로 이어져있어도 선택)
E[attr^="val"]	"attr"속성 값이 "val"로 시작하는 요소 선택
E[attr\$="val"]	"attr"속성 값이 "val"로 끝나는 요소 선택
E[attr*="val"]	"attr"속성 값에 "val"을 조금이라도 포함하는 요소 선택 (단어로 구성되지 않아도 선택)
E[attr =“val”]	"attr"속성 값이 정확하게 "val"이거나 "val-"로 시작하는 요소 선택

## 가상(pseudo) 선택자

웹 문서 소스에 실재하지 않는 요소나 필요에 의해 임의로 가상의 선택자를 지정하여 사용하는 것이다. 이벤트 기반으로 선택을 하거나 특정 순번에 의해 선택할 수 있다

### - 가상(pseudo) 클래스 선택자

선택자 패턴	의미
E:link	방문 전 링크를 선택
E:visited	방문 후 링크를 선택
E:active	마우스를 클릭할 때 선택
E:hover	마우스를 요소에 올라가 있는 동안 선택
E:focus	요소에 포커스가 머물러 있는 동안 선택

:link, :visited 선택자는 <a> 태그 요소와 관련이 있다. 사용자가 링크를 방문한 적이 있는지 없는지에 따라 선택될 지가 결정된다.

- 가상(pseudo) 요소 선택자

선택자 패턴	의미
E:root	문서의 최상위 요소(html)를 선택
Enth-child(n)	앞으로부터 지정된 순서와 일치하는 요소 E 선택 (E가 아닌 요소가 계산에 포함된다)
Enth-last-child(n)	뒤로부터 지정된 순서와 일치하는 요소 E 선택 (E가 아닌 요소의 계산에 포함된다)
Enth-of-type(n)	E 요소 중 앞으로부터 순서가 일치하는 E 요소 선택 (E 요소의 순서만 계산에 포함)
Enth-last-of-type(n)	E 요소 중 끝으로부터 순서가 일치하는 E 요소 선택 (E 요소의 순서만 계산에 포함)
E:first-child	첫 번째 등장하는 요소가 E 라면 선택 (E가 아닌 요소의 순서가 계산에 포함된다)
E:last-child	마지막에 등장하는 요소가 E 라면 선택 (E가 아닌 요소의 순서가 계산에 포함된다)
E:first-of-type	E 요소 중 첫 번째 E를 선택 (E 요소의 순서만 계산에 포함)
E:last-of-type	E 요소 중 마지막 E를 선택 (E 요소의 순서만 계산에 포함)
E:only-child	E 요소가 유일한 자식이면 선택 (E가 아닌 요소가 하나라도 포함되면 선택하지 않습니다.)
E:only-of-type	E 요소가 유일한 타입이면 선택 (E 아닌 요소가 포함되어도 E 타입이 유일하면 선택)
E:empty	텍스트 및 공백을 포함하여 자식 요소가 없는 E를 선택

**부정 선택자**

선택자 패턴	의미
E:not(S)	S가 아닌 E 요소 선택

조건을 주어 만족하지 않는 요소를 선택한다

- 부정 선택자 예제

<pre>&lt;html&gt; &lt;head&gt; &lt;style&gt;</pre>
--

```

/* CSS */
p:not(.class1) { color: red; }
</style>
</head>
<body>
<!--HTML -->
<p class="class1">선택되지 않음</p>
<p>부정 선택자</p>
</body>
</html>

```

클래스 속성을 "class1"으로 가지지 않는 <p> 태그 요소만 선택

## 5.2 선택자 우선순위

기본적으로 선언된 순서에 따라 적용되지만 선택자와 CSS 적용 방법에 따라 우선순위가 달라진다. 적용방법이 같다면 선택자의 종류와 수에 따라 우선순위가 결정된다. 이 때 높은 우선순위를 가진 선택자를 많이 사용할수록 해당 스타일이 우선적으로 적용된다.

- 선택자 우선 순위
  1. !important
  2. 인라인 스타일
  3. 아이디 선택자
  4. 클래스/속성/가상 선택자
  5. 태그 선택자
  6. 전체 선택자
- 삽입 위치에 따른 우선 순위
  1. <head> 요소 안의 임베디드 방식으로 적용
  2. <style> 요소 안의 @import 문
  3. <link> 요소로 연결된 CSS 파일
  4. <link> 요소로 연결된 CSS 파일 안의 @import 문
  5. 브라우저의 기본 스타일 시트

CSS를 적용하는 방법이 다양해질수록 그 우선순위를 파악하기 힘들어질 수 있다. 때문에 연결되는 CSS파일의 수를 최소화 해야 하며 @import문의 사용을 줄여야 한다.

## !important

!important는 스타일 구문 끝에 추가하여 설정할 수 있는데 우선순위를 1순위로 만들어준다

### - !important 예제

```
<html>
<head>
<style>
/* CSS */
.red { color: red !important; }
#p1 { color: black; }
</style>
</head>
<body>
<!--HTML -->
<p class="red" id="p1">!important 적용</p>
</body>
</html>
```

아이디 선택자는 클래스 선택자보다 우선순위가 높으므로 !important가 없다면 아이디 선택자 #p1의 스타일이 우선 적용됐을 것이다. 하지만 !important는 모든 우선순위를 무시하고 절대적으로 1순위를 만들어 주므로 .red 선택자가 적용된다.

## 6. CSS 주석

CSS 구문에 주석을 사용해 코드의 설명을 추가할 수 있다. 선택자의 역할, 스타일 구문의 의미 등을 주석으로 달아 디자이너-개발자 간의 소통을 원활하게 할 수도 있으며 유지보수도 용이해지게 된다.

### - CSS 주석

```
/* 주석 문장 */
```

## 7. 텍스트 관련

HTML 문서에는 텍스트가 상당히 많은 비중을 차지하며 CSS를 사용해 텍스트의 스타일을 지정하는 일이 많다

### 7.1 font-size

텍스트의 크기를 설정하는 속성

```
<html>
<head> </head>
<body>
<p style="font-size: 14px">14픽셀크기 </p>
</body>
</html>
```

### 7.2 color

텍스트의 색상을 지정

<p> 태그는 물론이며 <a> 태그, <span> 태그, <td> 태그 등 여러 태그에 적용할 수 있다

```
<html>
<head> </head>
<body>
<p style="color: red;">붉은색 </p>
</body>
</html>
```

### 7.3 font-style

텍스트의 스타일을 주는 속성

주로 이탤릭체로 설정할 때 사용한다

normal(기본), italic(이탤릭), oblique(기울임꼴)을 설정할 수 있다

```
<style>
p {
    font-style: italic;
}
</style>
```



## 7.4 font-weight

텍스트의 굵기를 조절하는 속성

normal, bold, bolder, lighter 사용 가능

100(얇음) ~ 900(두꺼움) 의 백단위 숫자 가능

(400 = normal, 700 = bold)

```
<style>
p {
    font-weight: bold;
    /* font-weight: 700; */
}
</style>
```

## 7.5 font-variant

대소문자에 대한 스타일

소문자를 작은 대문자 형태로 표현하도록 설정할 수 있다

```
<style>
p {
    font-variant: small-caps;
}
</style>
```

## 7.6 line-height

줄 간격을 지정

픽셀단위로 지정할 수 있지만 주로 %를 이용하여 지정을 하며 가독성과 관련이 있다

한글의 경우 100% 이상으로 지정해야 가독성이 좋아진다

단위 없이 '1.5'와 같이 숫자만 입력할 경우 em과 동일하게 인식한다(=150%)

```
<html>
<head> </head>
<body>
<p style="line-height:140%; width: 200px;">CSS is a language that describes the style of an HTML
```

```
document.</p>
</body>
</html>
```

## 7.7 font

텍스트의 스타일을 한번에 설정하는 속성

font-style, font-variant, font-weight, font-size/line-height, font-family 설정을 할 수 있다

모든 설정을 다 할 필요는 없고 적용하려는 스타일만 넣어주면 된다

단, 마지막의 font-family는 반드시 넣어주어야 한다

```
<style>
p {
  font-style: italic;
  font-weight: bold;
  font-size: 12px;
  line-height: 1.6;
  font-family: arial, - 68 -elvetica, sans-serif;

  /* 위의 속성들을 아래와 같이 선언할 수 있다 */
  font: italic bold 12px/1.6 arial, - 68 -elvetica, sans-serif;
}
</style>
```

## 7.8 text-align

텍스트의 정렬방향을 지정한다

left, right, center, justify 네 가지 속성값 가능

블록 요소에만 적용할 수 있는 속성이며 블록 요소 안의 텍스트 뿐만 아니라 인라인 요소도 같이 정렬된다

```
<html>
<head></head>
<body>
<p style="text-align: left">왼쪽 정렬</p>
<p style="text-align: right">오른쪽 정렬</p>
```

```

<p style="text-align: center">가운데 정렬</p>
<div style="text-align: center; ">  </div>
</body>
</html>

```

인라인 요소인 <img>를 가운데 정렬시키기 위해 블록 요소인 <div>를 이용

## 7.9 text-indent

들여쓰기 효과를 지정

문단의 첫 번째 줄을 지정한 길이만큼 들여쓰기 한다

```

<html>
<head> </head>
<body>
<p style="text-indent: 20px; width: 200px;">CSS is a language that describes the style of an HTML
document.</p>
</body>
</html>

```

## 7.10 text-decoration

텍스트 꾸밈 효과를 넣는 속성

밑줄, 윗줄, 취소선을 넣을 수 있다

```

<html>
<head> </head>
<body>
<p style="text-decoration: underline;">밑줄</p>
<p style="text-decoration: overline;">윗줄</p>
<p style="text-decoration: line-through;">취소선</p>
</body>
</html>

```

## 8. 레이아웃

### 8.1 width, height 속성

요소의 너비와 높이를 지정

- 가능한 속성 값
  - auto : 브라우저가 너비와 높이를 계산(기본값)
  - length : 소수점 숫자 뒤에 단위 지정자를 이용하여 값 지정
  - % : 너비와 높이를 상대적으로 적용

### max-height, max-width, min-height, min-width

요소의 최대 혹은 최소 너비와 높이를 지정

- 가능한 속성 값
  - none : 지정하지 않음(기본값)
  - length : 소수점 숫자 뒤에 단위 지정자를 이용하여 값 지정
  - % : 너비와 높이를 상대적으로 적용

### 8.2 CSS 여백속성 – margin, padding



#### margin

요소의 테두리 바깥쪽에 투명한 공간을 확보한다

auto, 길이, %로 속성값을 줄 수 있다

margin의 설정 값은 1개~4개로 줄 수 있다

```

<style>
div {
    /* margin-top: 10px, margin-right: 5px, margin-bottom: 15px, margin-right: 10px; */
    margin: 10px 5px 15px 10px;
    margin: 10px 5px 15px; /* top 10px, left&right 5px, bottom 15px */
    margin: 10px 5px; /* top&bottom 10px, left&right 5px */
    margin: 10px; /* all 10px */
}
</style>

```

## padding

요소의 테두리 안쪽과 콘텐츠와의 사이에 투명한 공간을 확보한다

auto, 길이, %로 속성값을 줄 수 있다

margin과 마찬가지로 설정 값은 1개~4개로 줄 수 있다

```

<style>
div {
    /* padding-top: 10px, padding-right: 5px, padding-bottom: 15px, padding-right: 10px; */
    padding: 10px 5px 15px 10px;
    padding: 10px 5px 15px; /* top 10px, left&right 5px, bottom 15px */
    padding: 10px 5px; /* top&bottom 10px, left&right 5px */
    padding: 10px; /* all 10px */
}
</style>

```

## 8.3 CSS 흐름속성 – float, clear

### float

박스의 배치 방향을 지정한다

절대 위치를 갖는 요소에는 적용되지 않는다

#### - 가능한 속성 값

left : 요소를 왼쪽으로 float 시킨다

right : 요소를 오른쪽으로 float 시킨다

none : 요소를 float 시키지 않는다

## clear

float에 의해 변화된 박스의 흐름을 원상태로 돌린다

float으로 흐름을 변경하면 이후의 요소들도 영향을 받으므로 clear 속성을 적절히 사용해야 한다

- 가능한 속성 값
  - left : 왼쪽으로 설정된 float을 제거
  - right : 오른쪽으로 설정된 float을 제거
  - both : 양쪽 float을 제거
  - none : 양쪽 흐름 모두 허락(기본값)

## 8.4 overflow와 Text-overflow 속성

### overflow

콘텐츠가 요소의 박스를 넘칠 때 표현방법을 지정

overflow-x(수평방향 컨트롤), overflow-y(수직방향 컨트롤) 속성에 대한 축약 속성이다

두 번째 값은 선택적이며, 값을 하나만 지정하면 overflow-x, overflow-y 속성 모두에 적용

- 가능한 속성 값
  - visible : 넘치는 콘텐츠를 자르지 않고 요소 박스를 넘어 표시 (기본값)
  - hidden : 넘치는 콘텐츠를 자르고 보이지 않게 한다
  - scroll : 넘치는 콘텐츠를 자르지만 스크롤바를 표시한다
  - auto : 넘치는 콘텐츠를 자르지만 스크롤바가 표시된다

### text-overflow

요소 내에 문자열의 넘침 현상을 처리하는 표현방법을 지정

- 가능한 속성 값
  - clip : 텍스트를 잘라낸다
  - ellipsis : 텍스트가 잘렸다는 것을 표현하기 위해 말줄임표(...)를 표시
  - string : 지정된 문자열을 출력

## 8.5 display, visibility 속성

### display

요소를 표시하는 방법을 지정

요소의 블록 모델을 바꿀 수 있는 속성이다

## visibility

요소를 보여주거나 숨기도록 하는 속성

- 가능한 속성 값

visible : 박스가 보여진다

hidden : 박스가 보이지 않지만 공간 확보 때문에 레이아웃에 영향을 미친다

collapse : hidden과 유사하며 테이블의 내부 객체에 적용

## 8.6 박스 모델

박스모델은 CSS에서 기본이 되는 디자인으로 요소를 어떤 방식으로 표현할지 정의한다

박스모델은 인라인(inline), 인라인블록(inline-block), 블록(block), 테이블(table), 절대위치(absolute), 플로트(float)가 있다

너비(width)와 높이(height) 속성은 요소의 내부 박스 크기를 설정하고, 내부 박스 둘레에는 패딩(padding)이 있고, 패딩 둘레에 테두리(border)가 있으며 테두리 둘레에 마진(margin)이 있다. 마진 둘레의 박스가 외부 박스가 된다.

패딩, 테두리, 마진이 커지면 외부 박스가 커지지만 내부 박스의 너비와 높이는 변하지 않는다

### 인라인(inline) 박스

인라인 요소, 정적 인라인 박스 등으로 부르며 인라인 박스는 부모 요소의 너비를 초과하면 새 행으로 자동 줄바꿈된다

- width, height 속성

너비와 높이가 내부 박스의 내용에 맞춰 자동으로 줄어들기 때문에 적용되지 않는다

- margin, line-height 속성

가로 마진은 작동하고 상하 마진은 작동하지 않는다. 줄 간격을 조절하려면 line-height를 이용해야 한다

- border, padding 속성

상단과 하단 테두리를 지정하면 줄 간격이 늘어나거나 인라인 요소의 세로 위치는 변하지 않고 패딩 위아래에 테두리가 표시된다

line-height에 따라 테두리가 앞뒤 줄의 테두리와 겹칠 수도 있다.

### 인라인 블록(inline-block) 박스

인라인과 유사하지만 블록 박스처럼 마진, 테두리, 패딩, 너비, 높이 속성이 적용된다

- width, height 속성  
요소의 너비, 높이를 지정한다. 인라인 요소에 display: inline-block을 지정하면 원하는 width와 height를 설정할 수 있게된다. 내용에 맞춰 자동 축소되게 하려면 width: auto와 height: auto를 지정한다. 부모 요소에 맞춰 자동 확대되게 하려면 width: 100%를 지정한다. 인라인 블록 요소를 자동 확대 시키면 블록 요소와 같아진다.
- margin 속성  
margin 속성이 인라인 블록 요소에는 다르게 적용된다. margin-top 속성에 양수 값을 지정하면 줄 간격이 늘어나고 음수 값을 지정하면 줄 간격이 줄어든다. margin-bottom 속성에 양수 값을 지정하면 요소가 올라가고 음수 값을 지정하면 요소가 내려진다. margin-left 속성에 양수 값을 지정하면 앞 요소와 떨어지고 음수 값을 지정하면 앞 요소에 가까워진다. margin-right 속성에 양수 값을 지정하면 다음 요소가 멀리 밀려나고 음수 값을 지정하면 다음 요소가 가까이 당겨진다.
- border, padding 속성  
테두리와 패딩을 지정하면 인라인 요소 외부 크기가 커진다. 요소 자체와 그 뒤의 내용이 오른쪽으로 밀리며, 요소가 위로 올라가고 그 요소가 들어 있는 줄 간격이 늘어난다.
- vertical-align 속성  
inline-block 요소는 vertical-align 속성의 영향을 받는다

## 블록(block) 박스

블록, 블록요소, 정적 블록 박스 등으로 불린다

부모 요소의 너비와 높이에 맞춰 자동 확대되게 할 수도 있고, 부모 요소보다 작거나 큰 크기를 지정할 수도 있다. 부모 요소보다 큰 크기가 지정되면 부모요소 밖으로 넘쳐서 표시된다.

overflow 속성으로 넘치는 부분에 대한 처리를 결정할 수 있다

<hn>, <p>, <blockquote>, <dt>, <address>, <caption>은 종결 블록 요소로 내용이나 인라인 요소를 포함할 수 있지만 블록요소를 포함할 수 없는 요소이다

- width 속성  
블록 요소의 너비 지정  
기본값은 width: auto이며 부모 요소의 너비에 맞춰 늘어난다.
- height 속성  
블록 요소의 높이 지정  
기본값은 height: auto이며 모든 자식 블록 요소의 높이에 맞춰 줄어든다.
- margin-left, margin-right 속성  
요소의 좌우를 들어넣거나 내어써서 좌우에 여백을 넣는다  
가로 자동 축소가 적용되지 않는다.



- `margin-left: auto, margin-right: auto` 속성  
지정 크기 블록 요소의 가로 위치 정렬을 설정한다. `width` 속성에 크기를 지정해서 블록 요소의 크기를 조절할 경우, `margin-right: auto`를 지정하면 부모 요소의 왼쪽에 정렬되고 `margin-left: auto`를 지정하면 부모 요소의 오른쪽에 정렬된다. `margin-left: auto`와 `margin-right: auto`를 함께 지정하면 블록 요소는 부모 요소의 중앙에 정렬된다. (= `margin: 0 auto`)
- `margin-top, margin-bottom` 속성  
양수 값을 지정하면 블록 요소 간의 거리가 멀어지고 음수 값을 지정하면 가까워진다(겹쳐질 수 있음)
- `border, padding` 속성  
박스의 바깥쪽 영역이 늘어나고 블록 요소와 그 뒤의 블록들이 아래로 밀린다. 자동 확대 블록 요소에 좌우 테두리와 패딩을 지정하면 내부 박스의 너비가 줄어들고, 지정 크기 블록 요소에 좌우 테두리와 패딩을 지정하면 테두리와 패딩이 차지하는 영역으로 인해 내부 박스가 있을 공간이 부족해 아래로 밀린다.

## 테이블(table) 박스

셀로 구성된 행이 들어 있는 블록 박스

블록과 유사하며 내부의 셀은 행과 열의 테이블의 동작을 따른다

테이블에는 마진이 있고 패딩이 없으며 셀에는 패딩만 있고 마진이 없다

테이블과 셀의 테두리를 하나로 합쳐 표시할 경우 `<table>` 요소에 `"border-collapse: collapse;"` 속성을 지정해야 한다

## 절대 위치(absolute) 박스

원래 위치를 벗어나 위/아래에 레이아웃을 생성하고 가장 가깝게 배치된 부모 요소를 기준으로 배치되거나 화면에 고정되어 배치된다

정해진 크기를 지정할 수 있고, 내용에 맞춰 자동 축소되게 할 수 있으며 가장 가까운 부모 요소에 맞춰 확대되게 할 수도 있다. 모든 요소를 절대 위치를 지정할 수 있으며 다른 박스의 배치에 영향을 주지 않게 된다

## 플로트(float) 박스

플로트 속성이 지정된 요소는 원래 위치를 벗어나서 인접한 블록 요소의 테두리와 배경 위를 덮는다. 플로트 지정된 요소를 포함하던 부모 요소는 크기가 줄어들게 된다. 자식 요소가 모두 플로트 박스가 되면 부모 요소는 아예 내용이 없는 것과 같아진다. 플로트 요소는 원래 위치에서 벗어나지만 인접한 내용을 뒤로 밀어낸다. 플로트 요소는 원래 위치에서와 같은 세로 위치에 배치되며, 부모 요소의 왼쪽이나 오른쪽 패딩 영역에 가로로 배치된다. 플로트 요소는 같은 세로 지점의 다른 플로트 요소 옆에 쌓이며, 다른 플로트 요소 옆에 배치될 공간이 없으면 아래로 내려간다.

## 9. 테두리

### 9.1 border

테두리를 설정하는 속성

border-width, border-style, border-color 순으로 작성한다

몇몇 속성을 생략해도 상관없다

```
<style>
p {
    border: 5px solid red;
}
</style>
```

#### border-width

테두리의 두께를 지정한다

px단위로 많이 지정하며, thin, thick 과 같은 단어로 설정 가능하다

단어로 설정가능한 두께는 thin, thick, medium(기본값)이 있다

#### border-style

테두리 스타일을 지정한다

기본값은 solid이며, none, dashed, dotted, double, groove, ridge, inset, outset, hidden을 설정할 수 있다

No border.

A dotted border.

A dashed border.

A solid border.

A double border.

A groove border.

A ridge border.

An inset border.

An outset border.

A hidden border.

## **border-color**

테두리 색상을 지정한다

## **9.2 border-radius**

CSS3에 추가된 속성이며 테두리 모서리에 반영될 반지름을 지정한다

지정된 반지름에 따라 둥근 모서리 테두리가 된다

한 개만 설정하면 네 모서리가 똑같이 반영되고, 따로 적을 경우 top-left, top-right, bottom-right, bottom-left 순서로 값을 설정한다

```
<style>
h1 {
    border: 2px solid #ddd;
    border-radius: 25px;
}

h2 {
    border: 2px solid #ddd;
    border-radius: 2em 3em 4em 5em;
}
</style>
```

## **9.3 border-collapse**

표의 칸 종류를 조정하는 속성

표의 테두리가 중복되는 부분을 축소할 지 분리할 지 결정하는 속성

collapse, separate 속성을 줄 수 있다

```
<style>
table {
    border-collapse: collapse;
    /* border-collapse: separate; (기본값) */
}
</style>
```

## 10. CSS3에 추가된 속성

### vender-prefix(벤더프리픽스)

브라우저별로 따로 놓던 CSS3의 속성을 잡아주기 위해서 사용되기 시작된 프리픽스는 마크업시 Css의 Class앞에 -moz-, -webkit-, -o-, -ms-라는 각 브라우저에서 판독이 가능한 접두어를 붙여서 해당 브라우저에서 인식할 수 있게 하는 것을 지칭한다. 적혀있는 순서대로 읽고, 각브라우저별로 적합한 부분과 만나면 판단을 내리고 다음 속성으로 넘어간다.

vender-prefix	브라우저
-webkit-border-radius: 10px;	사파리, 크롬
-moz-border-radius: 10px;	파이어폭스
-o-border-radius: 10px;	오페라
-ms-border-radius: 10px;	인터넷 익스플로러
-ms-border-radius: 10px;	표준 속성

make1.html실습을 통해 자바스크립트 없이 구동되는 아쿠아버튼, 그림자버튼, 마우스 이벤트에 반응하는 css를 활용해보자.

make1.html 을 작성하자.

```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
<title>html5! css3!</title>
<link rel="stylesheet" href="make1.css">
</head>
<body>
</body>
</html>
```

같은 경로에 make1.css 를 작성하자.

```
@charset "UTF-8";
html, body, h1, h2, h3, h4, h5, h6, section, header, aside, footer, nav, article, figure,
```

```
fieldset, p, ul, ol, li, dl, dt, dd, form {
    margin: 0;padding: 0;
}
body {
    font: .8em Arial, sans-serif;
}
html {
    background: #ddd
}
```

make1.html 에 div 엘리먼트를 다음과 같이 추가하자.

```
<body>
  <div class="mission_a">
    <h1><A HREF="#">웹표준 2.0</A></h1>
  <div>
</body>
```

h1태그 하위의 a태그에 기본값으로 설정된 밑줄을 없애보자. make1.css에서 작업한다.

```
.mission_a h1 a {text-decoration: none;}
```

h1태그 하위의 a태그에 마우스오버시 박스를 출력해보자. 지정태그의 다음에 요소들을 출력할 수 있는 ::after를 사용하자. (지정태그의 이전에 요소를 출력하고자 한다면, ::before를 사용한다.)

```
.mission_a h1 a:hover::after {
    content:'HTML5! CSS3! 실습입니다';
    font: .7em Helvetica sans-serif;
    color:#fff;
}
```

**웹표준 2.0**HTML5! CSS3! 실습입니다.

임의의 문자를 <A HREF="#">웹표준 2.0</A>다음에 추가한후 확인해보면, content가 가진 공간때

**웹표준 2.0**HTML5! CSS3! 실습입니다.**테스트**

문에 해당문자가 밀리는 것을 확인할 수 있다.

```
<h1><A HREF="#">웹표준 2.0</A>테스트</h1>
```

임의의 문구는 다시 삭제하도록 한다.

content의 영역을 position: absolute로 설정하고, make1css 에 content의 스타일속성을 추가한다.

```
.mission_a h1 a:hover::after {  
    ....  
  
    position: absolute;  
  
    display: block;  
    margin-left: 120px;  
    top: 0;  
    padding: 10px;  
    background: #222;  
    border: 1px solid #555;  
}
```

**웹표준 2.0** HTML5! CSS3! 실습입니다.

make1.css에 border-radius와 box-shadow속성을 추가하자. 지원하는 브라우저에 적합한 처리를 위해서 벤터 프리픽스를 추가한다.

```
.mission_a h1 a:hover::after {  
    ....  
    -webkit-border-radius: 8px;  
    -moz-border-radius: 8px;  
    -ms-border-radius: 8px;  
    -o-border-radius: 8px;  
    border-radius: 8px;  
  
    -webkit-box-shadow: #ddd 0 0 10px, #ccc 0 0px 5px inset;  
    -moz-box-shadow: #ddd 0 0 10px, #ccc 0 0px 5px inset;  
    -ms-box-shadow: #ddd 0 0 10px, #ccc 0 0px 5px inset;  
    -o-box-shadow: #ddd 0 0 10px, #ccc 0 0px 5px inset;  
    box-shadow: #ddd 0 0 10px, #ccc 0 0px 5px inset;  
}
```

## 웹표준 2.0 HTML5! CSS3! 실습입니다.

make1.html에 새 요소를 추가하자. 이미지주소는 같은 경로의 images/qr.jpg이다.

```
<body>  
  <div class="mission_b">  
      
  </div>  
</body>
```

make1.css 해당 요소에 대한 스타일을 추가하자.

```
.mission_b {  
    position: relative;  
    top: 10px; left: 10px;  
    border: 1px solid #ccc;  
    width: 70px; height: 70px;  
}
```

해당이미지의 투명도를 0.3으로 설정하고, 마우스오버시 0.9로 transition처리를 해보자

```
.mission_b img{
  opacity : .3;
  -webkit-transition: opacity .3s ease-out;
  -moz-transition: opacity .3s ease-out;
  -ms-transition: opacity .3s ease-out;
  -o-transition: opacity .3s ease-out;
  transition: opacity .3s ease-out;
}

.mission_b img:hover{
  opacity: .9
}
```



슈도 셀렉터 :hover를 설정하기 전까지 아무런 일도 일어나지 않지만, 변경된 스타일(opacity)이 감지되면, transition속성이 동작한다. 기존 img태그의 스타일 속성에서 변경된 지점까지 0.3초의 시간 동안 opacity속성이 .3에서 .9로, ease-out 타이밍 속성에 따라 변화가 일어난다. hover이벤트가 끝나도 같은 방식으로 원래의 스타일 속성으로 돌아온다.

만약, 두가지 이상의 속성을 변경하고 싶다면, 콤마를 구분자로 나열하면 된다.



make1.html에 다음 메뉴 요소를 추가하자.

```
<body>
  <div class="mission_c">
    <ul>
      <li class="ab"><a href="#">HTML5 and CSS3</a></li>
      <li class="ac"><a href="#">HTML5 and CSS3</a></li>
      <li class="ad"><a href="#">HTML5 and CSS3</a></li>
      <li class="ae"><a href="#">HTML5 and CSS3</a></li>
      <li class="af"><a href="#">HTML5 and CSS3</a></li>
      <li class="ag"><a href="#">HTML5 and CSS3</a></li>
    </ul>
  </div>
</body>
```

make1.css에 다음 스타일을 추가하자.

```
.mission_c ul {
  position: relative;
  top: 10px;
  left: 10px;
}
.mission_c li {
  list-style: none;
}
```



또 다음과 같이 li 태그하위의 a태그의 스타일을 지정하자

```
.mission_c li a {  
    color: #000;  
    font-size: 2em;  
    font-weight: normal;  
    display: block;  
    height: 50px;  
    margin: 10px 0;  
    padding: 10px;  
    text-decoration: none;  
}
```

마우스의 반응 영역지정을 위해 display: block으로 설정한다.

## 웹표준 2.0



HTML5 and CSS3

HTML5 and CSS3

HTML5 and CSS3

HTML5 and CSS3

HTML5 and CSS3

HTML5 and CSS3

또 make1.css에서 .mission\_c li a태그에 text-shadow속성을 지정한다. text-shadow의 속성값의 순서는 차례로 수평그림자값 수직그림자값 번짐값 컬러값이다. 이때 컬러값의 위치는 맨앞 또는 맨뒤에 올수 있다.

## 웹표준 2.0



HTML5 and  
CSS3

HTML5 and  
CSS3

HTML5 and  
CSS3

HTML5 and  
CSS3

HTML5 and  
CSS3

HTML5 and  
CSS3

```
.mission_c li a {  
    ...  
  
    width: 200px;  
    -webkit-text-shadow: rgba(255,255,255, .5) 1px 1px 2px;  
    -moz-text-shadow: rgba(255,255,255, .5) 1px 1px 2px;  
    -ms-text-shadow: rgba(255,255,255, .5) 1px 1px 2px;  
    -o-text-shadow: rgba(255,255,255, .5) 1px 1px 2px;  
    text-shadow: rgba(255,255,255, .5) 1px 1px 2px;  
}
```

li 하위의 a 태그에 마우스오버 이벤트시 border 둥글림처리를 위한 속성 및 transition 관련 속성을 추가한다.

```
.mission_c li a:hover {  
    color: #FFFF00;  
    .mission_c li a:hover {  
        text-shadow: rgba(0,0,0, .7) 1px 1px 1px;  
        width: 230px;  
  
        -webkit-box-shadow: rgba(0,0,0, .3) 3px 3px 5px;  
        -moz-box-shadow: rgba(0,0,0, .3) 3px 3px 5px;  
        -ms-box-shadow: rgba(0,0,0, .3) 3px 3px 5px;  
        -o-box-shadow: rgba(0,0,0, .3) 3px 3px 5px;  
        box-shadow: rgba(0,0,0, .3) 3px 3px 5px;  
  
        -webkit-transition: width .3s ease-out;  
        -moz-transition: width .3s ease-out;  
        -ms-transition: width .3s ease-out;  
        -o-transition: width .3s ease-out;  
        transition: width .3s ease-out;  
    }  
}
```

#### 웹표준 2.0



HTML5 and  
CSS3

HTML5 and  
CSS3

HTML5 and CSS3

HTML5 and  
CSS3

HTML5 and  
CSS3

HTML5 and  
CSS3

make1.css에 마지막으로 리스트별 배경색깔 진하기 차이를 만드는 스타일을 추가하자.

```
.mission_c li.ab a:hover {background:rgba(153,0,204, .1);}  
.mission_c li.ac a:hover {background:rgba(153,0,204, .2);}  
.mission_c li.ad a:hover {background:rgba(153,0,204, .3);}  
.mission_c li.ae a:hover {background:rgba(153,0,204, .4);}  
.mission_c li.af a:hover {background:rgba(153,0,204, .5);}  
.mission_c li.ag a:hover {background:rgba(153,0,204, .6);}
```

## 웹표준 2.0



HTML5 and  
CSS3

HTML5 and  
CSS3

HTML5 and  
CSS3

HTML5 and  
CSS3

HTML5 and  
CSS3

HTML5 and CSS3

## **Chapter 3. 다양한 기기에 반응하는 반응형 웹 제작**

## 1. 반응형 웹이란?

반응형 웹 디자인을 기반으로 다양한 디바이스(PC, 태블릿 PC, 스마트폰, 스마트 TV 등)를 대응하는 웹을 말한다.

반응형 웹 (Responsive Web)은 기기의 해상도에 따라 레이아웃이 반응하여 웹페이지를 나타낸다.

하나의 소스로 다양한 디바이스의 해상도에 따라 다양한 레이아웃으로 표현할 수 있다.

## 2. 반응형 웹으로 제작하는 이유

### 한 번의 구축으로 다양한 디바이스 대응

예전에는 PC와 모바일(스마트폰)용 홈페이지를 각각 구축했었다. 하지만 이제는 2 가지만으로 구축하기에는 디바이스(장치, 기기)가 너무 다양해졌다. 스마트폰 안에서도 브랜드에 따라 해상도가 모두 다르다. 그래서 점점 더 반응형 웹의 선호도가 높아지고 있다.

### 유지/관리가 효율적이다.

1 개의 소스로 구축하기 때문에 콘텐츠 관리 또한 1 개로 가능하다. 모바일용, PC용 콘텐츠를 따로 생성하거나 수정할 필요가 없기 때문에 관리의 일원화로 매우 편리해진다. 홈페이지 방문자 또한 PC에서 보던 내용을 모바일로도 같은 내용을 볼 수 있는 장점이 있다.

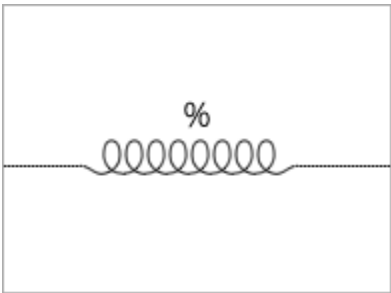
### 웹표준, 호환성 유지

반응형 웹으로 구축하려면 반드시 웹 표준을 지켜야 한다. 웹 표준을 따르면 전세계 다양한 웹브라우저(익스플로러, 크롬, 파이어폭스, 사파리, 오페라 등)의 호환성을 보장할 수 있다. 반응형 웹 디자인 및 개발이 일반 개발보다 쉽지는 않지만 워드프레스 같은 툴은 반응형 프리미엄 테마가 많아 유리합니다. 워드프레스는 관리자 페이지까지 반응형 웹으로 업그레이드 되어 있다. 그래서 PC가 아니더라도 간단한 홈페이지 수정 및 게시물 관리를 스마트폰으로도 할 수 있다.

모바일 콘텐츠 배려의 시대

현재 모바일 콘텐츠(페이지,글) 소비가 이미 PC를 넘어섰다. 즉, PC보다 모바일 기기(태블릿,스마트폰 등)로 더 많이 오래 접속한다는 뜻이다. 이제 콘텐츠의 생성 또한 모바일 사용자를 좀 더 배려하는 전략이 필요한 때이다.

1. 가변폭 설정하기



반응형 웹의 첫 번째 단계는 웹 문서의 너비를 가변폭으로 설정하는 것입니다. 웹 문서의 폭을 가변폭으로 설정하면 모든 해상도에 대응할 수 있습니다.

웹 문서 너비를 가변폭으로 설정하는 방법은 두 가지입니다.

- 1. 모든 구성 요소의 너비를 아예 지정하지 않는 방법
- 2. 모든 구성 요소의 너비를 가변폭으로 지정하는 방법

2. 최솟값/최댓값 설정하기





항상 웹 문서의 최소 너비와 최대 너비 값을 설정합니다.

예를 들어, 모든 단말(모바일, 태블릿, 데스크톱)의 해상도를 커버해야 한다면 320~1280 정도의 최소값과 최대값을 설정합니다.

태블릿과 데스크톱 대응 서비스를 만든다면 768~1280 정도를 설정합니다.

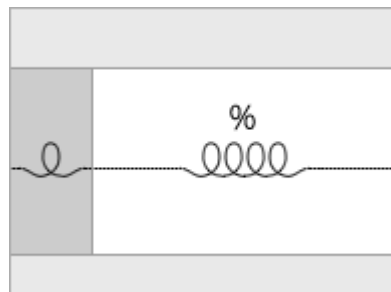
모바일과 태블릿 대응 서비스를 만든다면 320~1024 정도를 설정합니다.

최소값을 설정하지 않으면 웹 문서의 폭은 이론적으로 0px 이 될 수도 있습니다.

최대값을 설정하지 않으면 웹 문서의 폭은 수천 픽셀(또는 무한대)이 될 수도 있습니다.

이렇게 최소값과 최대값을 제한하는 이유는 우리의 예측이 언제든지 빗나갈 수 있다는 것(브라우저의 창 너비를 0px 에 가깝게 줄이거나 듀얼 모니터를 사용하는 경우)을 가정하기 때문입니다.

### 3. 컬럼 나누기



태블릿과 데스크톱 웹에서 대부분의 웹 문서가 2 개 이상의 컬럼을 사용하고 있습니다..

위 그림은 화면을 둘로 쪼갠 다음 각 컬럼에 20%, 80%에 해당하는 가변폭 너비를 설정해 준 경우입니다. 그러나 이런 구현은 다음과 같은 두 가지 문제가 있습니다.

**첫째,** 모든 컬럼의 너비가 문서의 최소/최대 너비에 의존하고 있다.

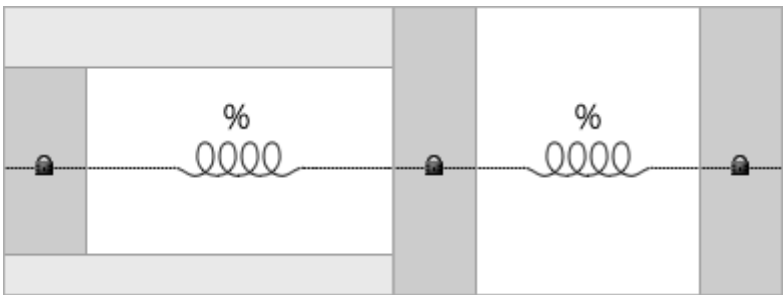
**둘째,** 모바일과 같이 좁은 화면에서는 컬럼 나누기가 적당하지 않다.

컬럼의 너비가 문서의 최소값/최대값 너비에 의존하고 있기 때문에 작은 폭의 컬럼은 작은 화면에서 너무 작게 표시됩니다. 예를 들어 화면 좌측의 네비게이션 영역은 보통 문자열의 길이가 달라지지 않기 때문에 고정폭으로 표시하는 것이 더 적절합니다.

네비게이션 영역을 가변폭으로 처리하게 되면 좁은 화면에서 모든 문자열을 표시하지 못하고 줄바꿈 될 것입니다.

모바일 단말의 스크린 사이즈는 최소 320px 까지 좁아집니다. 이런 작은 화면에서 컬럼을 나누는 것은 어렵습니다.. 컬럼 나누기 처리된 웹 문서를 모바일에서 적절하게 보여주기를 원한다면 미디어 쿼리 설정을 참고 하세요.

4. 하이브리드 레이아웃



반응형 웹에서 실제로 유용하게 쓰이는 컬럼 폭 설정 기법은 고정폭 컬럼과 가변폭 컬럼을 혼용해서 구현하는 하이브리드 레이아웃입니다.

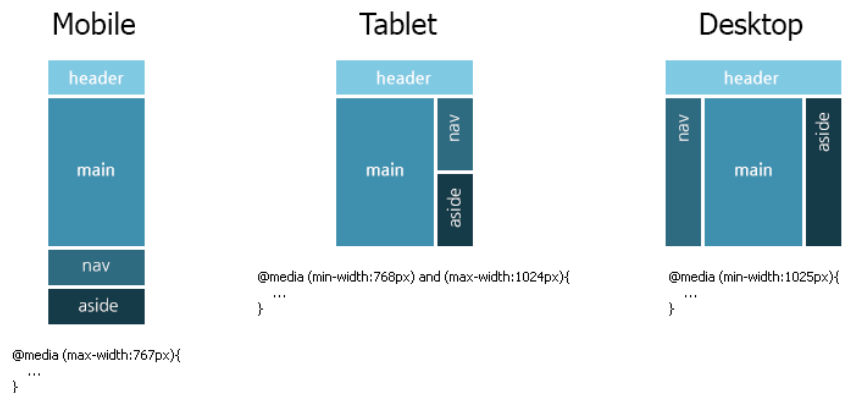
문서의 전체 너비와 본문 콘텐츠는 가변폭으로 설정했지만, 네비게이션 또는 사이드바 영역에는 독립적으로 고정폭을 적용함으로써 화면 폭과 무관하게 충분한 너비를 확보할 수 있게 됩니다.

**첫번째,** 하이브리드 2 컬럼 레이아웃 = 고정폭 + 가변폭

**두번째,** 하이브리드 3 컬럼 레이아웃 = 고정폭 + 가변폭 + 고정폭

하이브리드 레이아웃 기법으로는 데스크톱이나 테블릿 정도의 단말을 커버할 수 있을 것입니다. 그러나 컬럼 레이아웃을 사용하지 않는 모바일 화면에 대응할 수 없습니다. 모바일 단말에서 컬럼 레이아웃을 어떻게 처리해야 하는지 알고 싶다면 다음의 미디어 쿼리 설정 항목을 참고하세요.

5. 미디어 쿼리 설정



CSS3 미디어 쿼리는 화면 크기에 따라서 각기 다른 CSS 를 적용할 수 있는 중단점(혹은 분기점)을 제공합니다.

아래의 미디어 쿼리를 사용한 간단한 예제를 살펴보면, 데스크톱 브라우저의 창 크기를 조절해서 뷰 포트의 크기가 달라질 때 마다 서로 다른 내용과 스타일이 화면에 표시되게 작성한 것을 볼 수 있습니다.

```
/* All – 모든 해상도에서 해석하는 코드(미디어 쿼리 적용 안 함) */
...

/* Mobile – 767px 이하 해상도에서 해석하는 코드 */
@media (max-width:767px){ ... }

/* Tablet & Desktop – 768px 이상 해상도에서 해석하는 코드 */
@media (min-width:768px){ ... }

/* Tablet – 768px~1024px 해상도에서 해석하는 코드 */
@media (min-width:768px) and (max-width:1024px){ ... }

/* Desktop – 1025px 이상 해상도에서 해석하는 코드 */
@media (min-width:1025px){ ... }
```

1. 전체 폭이 가변폭으로 설정되어 있고,
2. 최소값과 최대값이 설정되어 있고,
3. 컬럼이 존재하며,

- 4. 고정폭과 가변폭이 혼합되어 있고,
- 5 미디어 쿼리를 사용해서 적절한 중단점을 제공했습니다.

하지만 아직 콘텐츠를 포함하기에 완벽한 것은 아닙니다. 데스크톱 브라우저의 창 크기를 조절하여 본문 영역에 포함된 이미지와 미디어 콘텐츠가 가변폭에 대응하도록 설정하여야 합니다.

## 6. 가변폭 미디어 콘텐츠

### Fixed content



### Flexible content



본문 영역을 가변폭으로 처리했다면 포함하는 콘텐츠 역시 가변폭에 대응해야 합니다.

일반적인 문자열은 가변폭을 가진 부모 요소의 너비에 따라서 자동으로 줄바꿈 처리가 되지만 이미지와 비디오 같은 미디어 콘텐츠 요소는 기본적으로 고정폭을 지니고 있기 때문에 가변폭을 가진 본문 너비에 대응할 수 있도록 너비를 다시 설정해 주어야 합니다.

다음과 같은 원칙을 적용할 수 있습니다.

- 본문 너비보다 작은 너비의 이미지는 아무런 처리를 하지 않는다.
- 본문 너비보다 큰 너비의 이미지는 본문 너비를 초과하지 않도록 너비를 100%로 재설정 한다.
- 비디오는 본문 너비와 동일하도록 무조건 100%로 다시 설정한다.
- 이미지 또는 비디오의 너비가 재설정 될 때 높이 또한 동일한 비율로 다시 설정한다.

이런 원칙을 적용하면 가변폭에 완벽하게 대응하는 본문 콘텐츠를 만들 수 있습니다. 데스크톱 브라우저의 창 크기를 조절하여 본문 콘텐츠(이미지, 비디오)가 가변폭에 대응하게 합니다.

적용한 소스코드는 다음과 같습니다.

```
img, video{ max-width:100%; height:auto; }  
video{ min-width:100%; }
```

## 7. 고정폭 테이블

반응형 웹에서 어려운 문제 중의 하나가 테이블 입니다. 스크린 사이즈가 넓은 단말에서는 테이블의 내용이 잘 보여지지만 작은 사이즈의 모바일 환경에서 표를 표현하는 것은 어렵습니다.

해결해 볼 수 있는 몇 가지 대안을 살펴보면

**첫번째** 대안은 CSS 만을 활용한 것인데 표 하나를 만들 때마다 로컬 CSS 코드를 추가해야 하기 때문에 번거롭습니다..

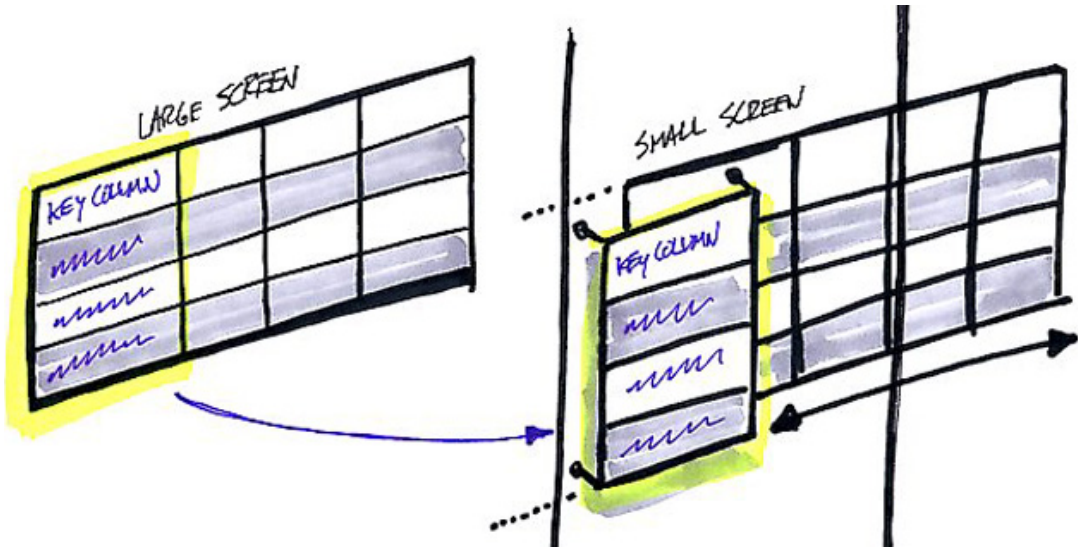
**두번째** 대안은 자바스크립트를 추가하여 반복 작업을 줄이는 방법입니다.

반응형 테이블을 표현하기를 원한다면 데스크톱 브라우저의 창 크기를 조절하여 좁은 화면에서 테이블이 어떻게 표현되는지 살펴 봅니다.

```
.table{ overflow:auto; }
```

```
<div class="table">  
<table>...</table>  
</div>
```

테이블이 본문 폭을 넘칠 때 가로로 스크롤 할 수 있도록 처리한 것입니다.



이미지 출처: A New Take on Responsive Tables.

## 8. 모바일 뷰 포트 설정

스마트폰 기기는 태블릿이나 데스크톱과 달리 독자적인 뷰 포트 처리 방식을 가지고 있습니다.

태블릿과 데스크톱의 뷰 포트는 웹 문서의 너비와 무관하게 화면에 보이는 영역이 뷰 포트가 되고, 데스크톱은 웹 브라우저의 창 크기를 조절함으로써 사용자가 뷰 포트의 크기를 직접 제어할 수도 있게 됩니다.

한편 스마트폰의 경우 단말의 스크린 사이즈와 무관하게 웹 문서의 너비와 높이가 뷰 포트가 되고 스마트폰 스크린에 웹 문서(뷰 포트)를 모두 출력하기 위해 스크린 사이즈보다 큰 문서는 자동으로 줌 아웃 처리를 합니다. 다시 말하면 뷰 포트를 스크린 크기에 맞게 강제로 축소하는 것입니다. 결과적으로 데스크톱에 대응하도록 만들어진 문서를 스마트폰으로 열어 보면 통상 3~4 배 이상 줌 아웃 되어 글씨를 알아 볼 수 없는 형태로 작게 표시합니다.

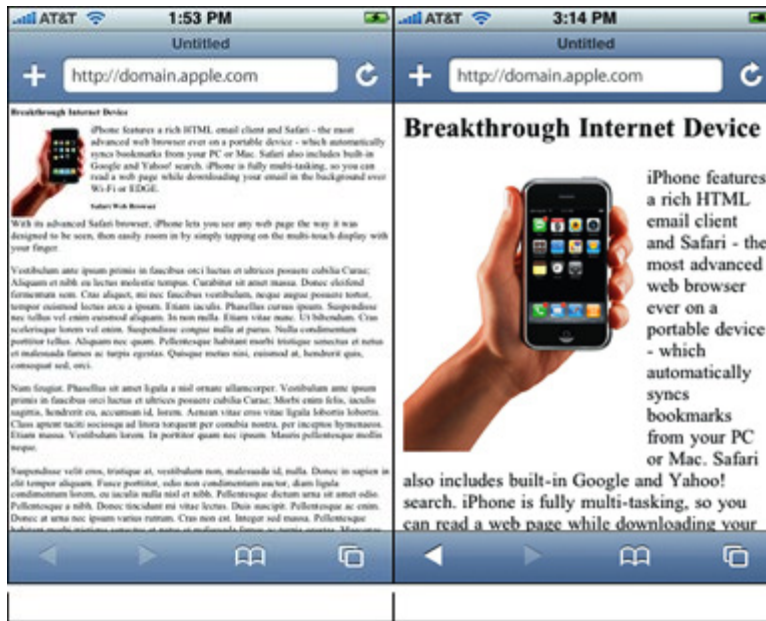
이 문제를 해결하려면 HTML 문서 <head> 영역에 모바일 뷰 포트를 재설정하는 <meta> 코드 한 줄을 추가하면 간단하게 해결 됩니다.

```
<meta name="viewport" content="width=device-width">
```

이 코드는 스마트폰 웹 브라우저에서만 해석이 되고, 스마트폰의 뷰 포트 영역은 더 이상 문서가 아니라 스크린 영역이 됩니다.

위 코드는 다음과 같이 작용합니다.

- 뷰 포트를 문서가 아닌 스크린 영역으로 재 설정.
- 웹 문서를 강제로 스크린 사이즈에 맞게 줌 아웃 하지 않음.



default = 980 pixels

320 pixels

모바일 뷰 포트 적용 전후 스마트폰 화면 표시 예.

이미지 출처: Configuring the viewport.

뷰 포트를 재 설정함으로써 스마트폰에서 웹 문서를 원본 크기로 표시하게 되면 스크린 사이즈보다 더 큰 너비를 가지고 있는 문서는 가로 스크롤을 유발하게 되는데, 웹 문서와 콘텐츠의 너비를 가변폭으로 설정하고 미디어 쿼리를 이용해서 좁은 스크린 상태의 레이아웃을 다시 배치하면 됩니다.

## 9. 글꼴 크기를 반응형으로 제어

hi hi hi  
hiihihihi  
hi



모바일 뷰 포트 설정을 통해서 화면 표시 배율을 적절하게 만들면 스마트폰에서 글꼴이 너무 작아지는 문제는 발생하지 않기 때문에 글꼴 크기까지 반응형으로 직접 제어해야 하는 것은 어렵습니다. 읽기에 적절한 크기로 표시하기 때문에 일반적으로 글꼴 크기까지 직접 제어할 필요는 없습니다

그러나 디자이너 의도에 따라 화면 폭에 알맞게 적절한 글자 수를 배치하고 동적인 화면 크기에 따라 동등한 배율로 글꼴 크기를 표현해야 할 때 개발자는 의도에 따라 제어할 수 있어야 합니다.

웹 브라우저의 창 크기를 조절하여 화면 크기와 글꼴 크기가 비례하여 조절이 되는지 확인해 봅니다

- 화면 크기와 글꼴 크기가 비례하지 않는 반응형 웹.
- 화면 크기와 글꼴 크기가 비례하는 반응형 웹 + 반응형 글꼴.

반응형 글꼴 크기의 비밀은 rem(root em) 이라 부르는 단위에 있습니다. HTML 에서 root 는 곧 `<html>` 요소를 의미하는데, 글꼴 크기에 rem 이라는 단위를 적용하면 `<html>` 요소의 글꼴 크기로부터 자신(`<body>`, `<div>`, `<span>` 등등...)의 글꼴 크기를 상대적으로 계산합니다. 상대적이라는 의미는 `<html>` 요소의 글꼴 크기가 바뀔 때 자신의 글꼴 크기도 함께 변경이 된다는 의미 입니다.

그렇다면 `<html>` 요소의 글꼴 크기는 미디어 쿼리를 이용하면 화면 크기에 따라 동적으로 결정할 수 있습니다.

반응형 웹 글꼴은 결국 다음과 같은 절차로 실행합니다.

1. 미디어 쿼리를 이용해서 화면 너비에 비례하도록 `<html>` 요소의 글꼴 크기를 절대 단위(px)로 지정한다.
2. `<body>`, `<div>`, `<span>` 등등... 반응형으로 글꼴 크기를 제어하고자 하는 요소를 선택하여 글꼴 크기를 상대(rem) 단위로 지정한다.
3. 사용자 환경에 따라 뷰 포트의 너비가 달라지면 `<html>` 요소의 글꼴 절대 크기도 화면 크기에 비례하여 달라진다.
4. `<html>` 요소의 글꼴 크기가 달라지면 `<body>`, `<div>`, `<span>` 등등... rem 단위를 적용한 요소의 글꼴 크기도 `<html>` 요소의 글꼴 크기에 비례하여 함께 달라진다.

사용자 단말의 화면 크기는 <html> 요소의 글꼴 크기를 결정하고, rem 단위를 사용한 요소들은 <html> 요소로부터 글꼴 크기를 결정하게 되어 화면 크기가 글꼴 크기에 영향을 미치는 원리입니다.

만약 <html> 요소로부터 글꼴 크기를 상속 받을 수 있는 rem이라는 개념의 단위가 없다면 글꼴 크기는 절대 값으로 결정이 되거나, 또는 오직 부모 요소로부터 상속 받을 수 있기 때문에 미디어 쿼리가 있다 하더라도 훨씬 더 복잡한 로직으로 구현할 수 밖에 없게 됩니다.

## 10. 웹 폰트 사용

Web fonts  
Web fonts  
Web fonts

웹 폰트를 사용하고 싶다면 미디어 쿼리를 통해 스마트폰 단말 영역으로 추정할 수 있는 해상도를 분기하고 스마트폰 단말에는 적용하지 않는 것이 좋습니다.

그 이유는 다음과 같습니다.

- 한글 글꼴이 포함된 웹 폰트는 용량을 아무리 최적화 하더라도 일반적으로 웹 문서 용량의 30% 이상을 차지하게 된다.
- 웹 폰트는 느리게 로딩되기 때문에 글꼴이 로딩되기 전까지 화면을 출력하지 않거나 또는 글꼴이 로딩된 이후 화면을 다시 출력하는 문제가 있다.

따라서 웹 폰트를 사용하고 싶다면 테블릿 또는 데스크탑 화면으로 추정할 수 있는 화면 크기에 한해 제한적으로 적용하는 것이 좋습니다. 스마트폰 단말은 일반적으로 작은 화면에 최적화된 시스템 글꼴을 제공하기 때문에 모바일 환경에서 웹 폰트를 사용하지 않는 것이 더 좋은 UX라고 말할 수 있습니다.

작은 화면에서 웹 폰트를 사용하지 않도록 처리한 반응형 웹 예제. 웹 브라우저의 창 크기를 조절하여 큰 화면과 작은 화면에서 글꼴을 다르게 표시하도록 해야 합니다. 한편 웹 폰트 사용을 권장할만한 상황도 있습니다. 웹 폰트는 글꼴 아웃라인 정보만 담아낼 수 있는 것은 아니기 때문에 다양한 형태의 그래픽 정보를 표현할 수 있고 벡터 방식으로 출력하기 때문에 확대 축소를 해도 비트맵처럼 계단 현상이 발생하지 않습니다. 또한 웹 문서 안에서 글꼴처럼 다루기 때문에 하나의 심볼은 크기 뿐만 아니라 색상 또한 자유롭게 변경하여 재 사용이 가능합니다.



이미지 출처: Why I make Glyphicons.

창 크기를 조절하면 이미지의 크기 또한 변경이 되는데 이 때 계단 현상없이 어떤 크기로도 또렷하게 표시할 수 있습니다.

- 웹 폰트는 글꼴 뿐만 아니라 벡터 그래픽을 표현할 수 있다.
- 하나의 심볼은 크기와 색상 및 투명도 등을 변경하여 재 사용할 수 있다.

웹 문서에 이미지 사용이 많은 경우 이미지 대신 웹 폰트 그래픽을 사용하면 이미지 전송 요청 횟수와 그래픽 파일 용량을 줄여서 웹 문서 로딩 성능을 개선할 수도 있습니다.

## **Chapter 4. JavaScript를 활용하여 다양한 기능 추가하기**

## 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 바탕으로 했기 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 요소

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	String 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.

<pre>&lt;script&gt;   console.log("hello".length); &lt;/script&gt;</pre>
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.

<pre>&lt;script&gt; console.log("hello world".charAt(0));    // 특정 인덱스 위치의 문자를 반환 &lt;/script&gt;</pre>
실행결과> h
<pre>&lt;script&gt; console.log("hello world".replace("hello", "hi"));    // 기존 문자열을 다른 문자열로 치환</pre>

<pre>&lt;/script&gt;</pre> <p>실행결과&gt; hi world</p>
<pre>&lt;script&gt; console.log("hello world".toUpperCase());           // 문자열을 대문자로 변환 &lt;/script&gt;</pre> <p>실행결과&gt; HELLO WORLD</p>
<pre>&lt;script&gt; console.log("1,2,3,4,5".split(",")); // ","를 토큰으로 문자열을 나누어 문자열 배열을 반환 &lt;/script&gt;</pre> <p>실행결과&gt; ["1", "2", "3", "4", "5"]</p>

## 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

<pre>&lt;script&gt; console.log(parseInt("010", 10));           // 두번째 인자로 진법을 지정한다 &lt;/script&gt;</pre> <p>실행결과&gt; 10      // 10진수로 "010"을 변환</p>
<pre>&lt;script&gt; console.log(parseInt("010", 2));           // 이진수로 지정 &lt;/script&gt;</pre> <p>실행결과&gt; 2</p>

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

<pre>&lt;script&gt;     console.log(parseInt("010"));           // 숫자 앞에 0이 있으면 8진수로 인식 &lt;/script&gt;</pre> <p>실행결과&gt; 8</p>
---

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>  
실행결과> 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.  
  }  
</script>
```

```

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 요소는 참이므로 이것이 실행된 것이다..."
    }

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

```

```

    }

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

```

존재하는 요소는 참이므로 이것이 실행된 것이다...

## 4 배열

배열이란 자료형이 일치하는 여러 개의 데이터를 나열하여 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);

```



<pre>//배열선언3 : 리터럴로 바로 초기화 var arr3 = ["x", "y", "z"]; console.log("arr3 = ", arr3); &lt;/script&gt;</pre>
<p>결과</p> <pre>arr1 = [a,b,c] arr2 = [l,m,n] arr3 = [x,y,z]</pre>

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

<pre>&lt;script&gt; var arrTest = []; arrTest[0] = "k"; arrTest.push("h"); console.log("arrTest= ["+arrTest+"]"); &lt;/script&gt;</pre>
<p>결과</p> <pre>arrTest= [k,h]</pre>

## 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. ( $0 \leq \text{index} < \text{배열크기}$ )

## 5 Object

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

자바스크립트를 객체기반 언어라고 하는 이유이기도 하다. 실질적으로 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 ,(coma)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

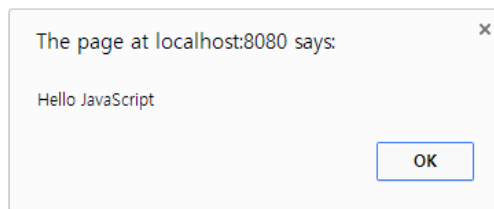
### alert() 메소드 사용

특정 정보를 메시지 창을 통해 사용자에게 알려주기 위해 사용한다.

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

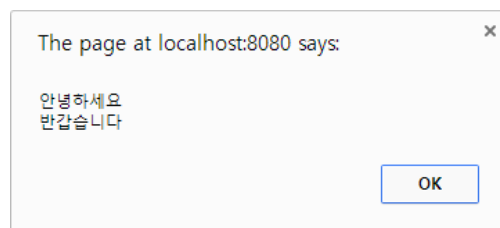
- 크롬브라우저에서 alert 확인

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



- 변수에 저장된 값 출력

```
<script>
    var txt = "안녕하세요. 반갑습니다.";
    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.

#### 비교연산자

상황	설명	예
숫자 == 문자열 비교	문자열을 숫자로 변환 후 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

#### 산술연산자

##### 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'+num1+' + ' + num2);
} else {    // num1 == num2 이 거짓일 때 실행
    alert('num1과 num2가 같지 않다'+num1+' + ' + 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 반복문이다.

### - 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

기본 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 역할을 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 선언했느냐에 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.

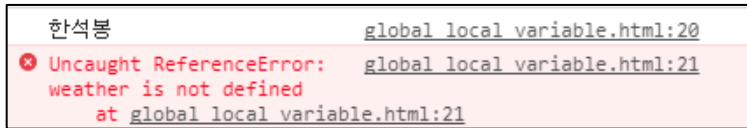
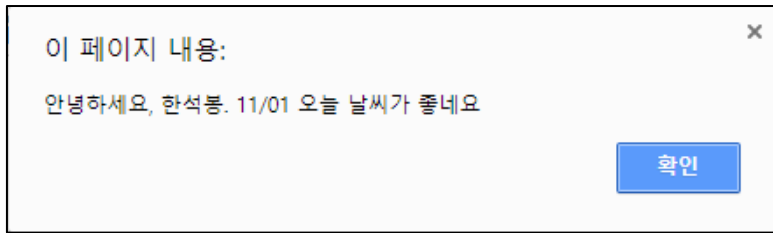
#### 지역변수 (Local Variables)

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.

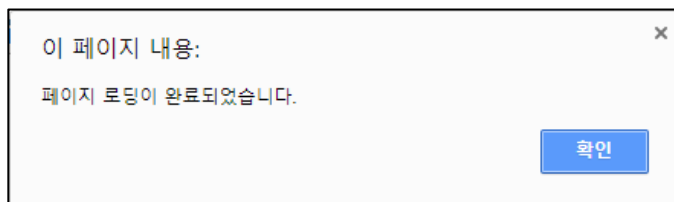
자바스크립트 함수를 사용 방법에 따라 분류하면 일반 함수, 중첩 함수, 콜백 함수로 분류할 수 있다.

### 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 addEventListener 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(e){
        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 엘리먼트객체
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])  
    }  
  }  
</script>
```

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

## 결과

```
isTrusted : true  
- 142 -imest : 483  
- 142 -imest : 138  
- 142 -imest : 44  
- 142 -imest : 22  
ctrlKey : false  
shiftKey : false  
altKey : false  
metaKey : false  
button : 0  
buttons : 0  
relatedTarget : null  
pageX : 44  
pageY : 22  
x : 44  
y : 22  
- 142 -imest : 34  
- 142 -imest : 12  
- 142 -imestam : 0  
- 142 -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
- 143 -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.

```

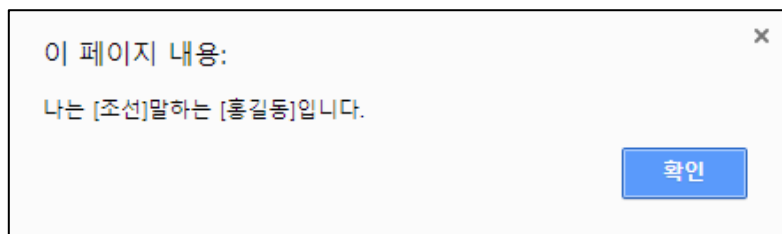
<script>
  var hong = {

```

```

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

```



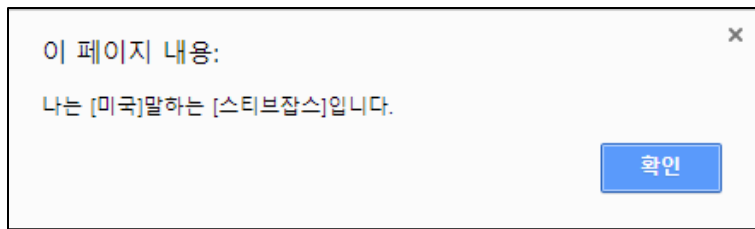
생성자 함수명은 관례에 따라 대문자로 시작하고, 객체 정보를 담고 있다. 자바에서 새로 객체를 생성하기 위해 해당 클래스의 생성자를 호출한 것과 닮아 있지만, 자바스크립트는 클래스 베이스의 객체지향 언어는 아니다.

```

<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.");
    }
  }

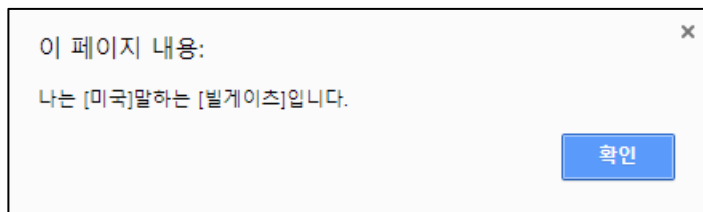
  //Developer생성자함수 선언
  function Developer(name, age, gender, nationality, job, exp){
    this.name = name;
    this.age = age;
    this.gender = gender;
    this.nationality = nationality;
    this.job = job;
    this.exp = exp;
    this.work = function(){
```

```

        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>

```



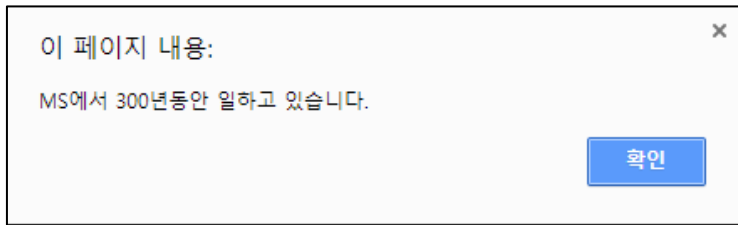
Developer의 프로토타입으로 매개변수 없는 Person 생성자를 전달하면서 Person 객체를 상속받고 있다. 직접 선언하지 않은 부모 객체의 속성에도 접근할 수 있다. 또한 Developer의 생성자 함수에서 부모 객체 Person 함수의 call 메소드를 사용할 수도 있다.

```

<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() {
  }
  inner();
}
</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 tlD = setInterval(function () {
    count++;
    alert(count);

    if(count == 5) {
        clearInterval(tlD);
    }
}, 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.round() 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 = "ABCDEFGH";
alert(alpha.charAt(3)); // D
</script>
```

## 특정 문자열을 원하는 문자열로 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, newSubstr|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) – 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);
```

<pre> console.log(name1); console.log(name2); console.log(name3); &lt;/script&gt; </pre>
홍길동 신사임당 유관순

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

Str.substr(start[, length])
<pre> &lt;script&gt; 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); &lt;/script&gt; </pre>
홍길동 신사임당 유관순

### 좌우공백 제거함수 – trim()

제거한후에는 새로운 문자열을 리턴한다. 기존 문자열은 변경하지 않는다

<pre> &lt;script&gt; var strtrim = "      white   Space      " var newstr = strtrim.trim(); console.log(newstr); &lt;/script&gt; </pre>
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

<pre>&lt;script&gt; var arr = [ "Apple", "Banana", "Cherry" ]; &lt;/script&gt;</pre>
--

## 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: f}
{x: "x 속성값", y: "y(문자열) 속성값", z: f}
x 속성값
x 속성값
y(문자열) 속성값
y(문자열) 속성값
f () {
    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 것도 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: "예뻐",
    breed: "푸들",
```

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

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

```
Console.log(dog2);
```

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

```
//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: "예뻐", breed: "푸들", bark: f}
```

```
예뻐
```

```
왈왈
```

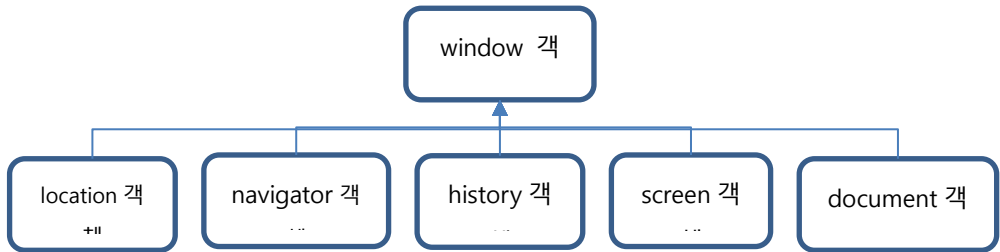
```
뽀뽀
```



## 13.2 자바스크립트 브라우저 객체 모델 (BOM)

BOM(Browser Object Model)이란 웹브라우저의 창이나 프레임을 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

```
ow.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
<b>_blank</b>	새창(기본값)
<b>_parent</b>	부모창
<b>_self</b>	현재창
<b>_top</b>	로딩가능한 아무 창

세번째 매개변수 : 원도를 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	브라우저 도구모음 유무	yes, no, 1, 0

네 번째 매개변수 : 히스토리 목록에 현재 문서를 대체 혹은 새 항목 여부

[illegible]

## window 객체의 기본 메서드

자산의 형태와 위치를 변경할 수 있는 메서드를 제공

\_\_By() 형태의 메서드는 현재 윈도우를 기준으로 상대적으로 속성을 변화

\_\_To() 형태의 메서드는 절대적인 기준으로 속성을 변화

메소드명	설명
<b>moveBy(x,y)</b>	윈도우 위치를 상대적으로 이동
<b>moveTo(x,y)</b>	윈도우 위치를 절대적으로 이동
<b>resizeBy(x,y)</b>	윈도우 크기를 상대적으로 지정
<b>resizeTo(x,y)</b>	윈도우 크기를 절대적으로 지정
<b>scrollBy(x,y)</b>	윈도우 스크롤 위치를 상대적으로 이동
<b>scrollTo(x,y)</b>	윈도우 스크롤 위치를 절대적으로 이동
<b>focus()</b>	윈도우에 초점 가지기
<b>blur()</b>	윈도우에 초점 제거
<b>close()</b>	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>
```

setInterval 함수를 이용해서, child 윈도우 객체를 1초에 우로 10픽셀, 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	웹브라우저 이름	파이어폭스, 크롬 => "Netscape"
appVersion	브라우저 버전	"5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.100 Safari/537.36"
userAgent	브라우저가 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	브라우저가 동작하는 운영체제에 대한 정보	"Win32"
cookieEnabled	브라우저가 cookie사용이 가능한지 여부	true/false
online	브라우저의 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 나타낼 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, DOM의 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.

DOM을 사용하기 위해 특별히 해야할 일은 없다. 각각의 브라우저는 자신만의 방법으로 DOM을 구현하였으며, 모든 웹 브라우저는 스크립트가 접근할 수 있는 웹 페이지를 만들기 위해 어느 정도의 DOM을 항상 사용한다.

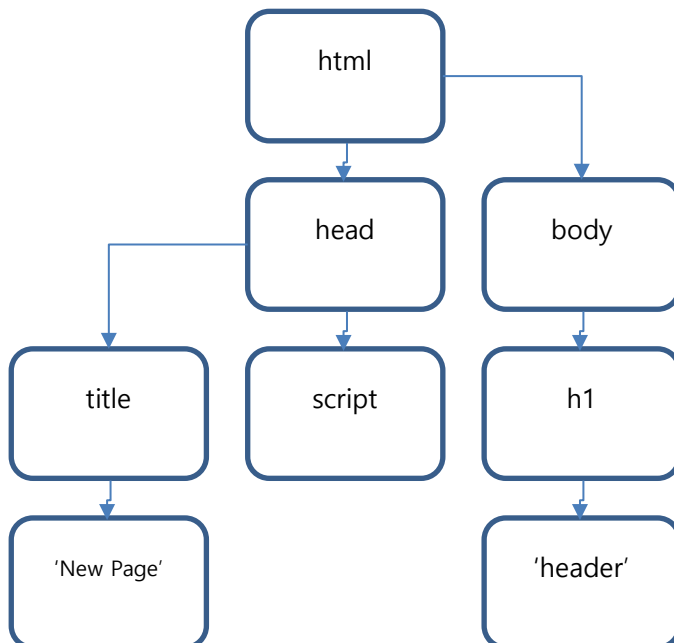
스크립트를 작성할 때 (인라인 <script> 요소를 사용하거나 웹 페이지 안에 있는 스크립트 로딩 명령을 사용하여), 문서 자체를 조작하거나 문서의 children 을 얻기 위해 document 또는 window elements 를 위한 API 를 즉시 사용할 수 있다.

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>, <div> 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>
```



- **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.innerHTML += 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

**JavaScript**  
**with DOM**

- 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 5. jQuery**

# 1. jQuery 개요

**jQuery** 는 2006년 존 레식(John Resig)에 의해서 소개되었으며, 클라이언트 측 솔루션 제작을 손쉽게 만들어주는 크로스 플랫폼 자바스크립트 라이브러리로 출발했다.

**jQuery**는 복잡한 자바스크립트 코드를 간단하고 조작하기 쉽게 구현할 수 있다는게 특징이며, 오늘날 가장 인기있는 자바스크립트 라이브러리이다.

**jQuery**는 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 브라우저를 지원하지만, 최신 버전은 그렇지 않다. 최신 버전을 사용하면서도, jQuery 구버전의 기능들을 지원하기 위해서 jQuery migrate 를 추가하면 된다.

이전 버전으로 작성된 코드를 계속 사용하거나, 구버전의 브라우저 지원을 위해 jQuery-migrate의 사용을 고려해 보자.

버전	대상
jQuery_Migrate 1.4.1	1.9 버전이하 코드를 1.9에서 3.0버전에서 사용가능하게 해줌.
jQuery_Migrate 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 읽어들이고 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>
    </ul>
</div>

```



```
<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>
```

## Front End

- html
- css
- JavaScript
  - JavaScript Core
  - DOM
  - BOM
- jQuery

## Back End

- java
- jsp/servlet
- oracle

### 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>
```

## Front End

- html
- css
- JavaScript
  - [JavaScript Core](#)
  - [DOM](#)
  - [BOM](#)
- jQuery

## Back End

- java
- jsp/servlet
- oracle

### 태그 이름으로 노드 찾기

```
$("#태그 이름")
```

태그 이름을 매개변수로 \$( )함수를 호출

### 태그 셀렉터 예제

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

## Front End

- html
- css
- JavaScript
  - [JavaScript Core](#)
  - [DOM](#)
  - [BOM](#)
- jQuery

## Back End

- java
- jsp/servlet
- oracle

### 속성 이름으로 노드 찾기

```
$("속성옵션")
```

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

<code>\$("E[A]")</code>	속성 A를 포함한 모든 E 노드
<code>\$("E[A=V]")</code>	속성 A의 값이 V인 모든 E 노드
<code>\$("E[A^=V]")</code>	속성 A의 값이 V로 시작하는 모든 E 노드
<code>\$("EA\$=V")</code>	속성 A의 값이 V로 끝나는 모든 E 노드
<code>\$("E[A*=V]")</code>	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>
```

## Front End

- html
- css
- JavaScript
  - [JavaScript Core](#)
  - [DOM](#)
  - [BOM](#)
- jQuery

## Back End

- java
- jsp/servlet
- oracle

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

## Front End

- html
- css
- JavaScript
  - [JavaScript Core](#)
    - DOM
    - BOM
- jQuery

## Back End

- java
- jsp/servlet
- oracle

### 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>
```

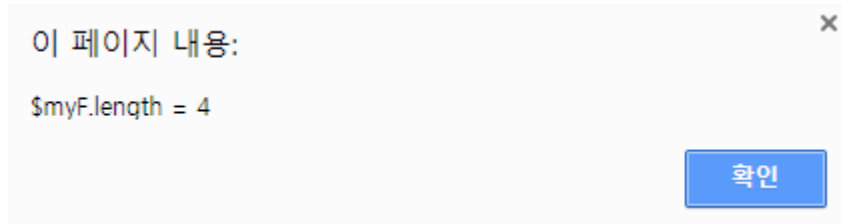
### 찾은 노드 개수 구하기 - `length` 프로퍼티

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>
```

jQuery는 각메서드가 해당객체를 리턴하기 때문에 연속적으로 메소드를 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>
```

## Front End

- html
- css
- JavaScript
  - [JavaScript Core](#)
  - [DOM](#)
  - [BOM](#)
- jQuery

## Back End

- java
- jsp/servlet
- oracle

### 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>
```

## Front End

- [html](#)
- [css](#)
- JavaScript
  - [JavaScript Core](#)
  - [DOM](#)
  - [BOM](#)
- jQuery

## Back End

- java
- jsp/servlet
- oracle

### 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>
```

## Front End

- html
- css
- JavaScript
  - [JavaScript Core](#)
  - [DOM](#)
  - [BOM](#)
- jQuery

## Back End

- java
- jsp/servlet
- oracle

### 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>
```

## Front End

- html
- css
- JavaScript
  - [JavaScript Core](#)
  - [DOM](#)
  - [BOM](#)
- jQuery

## Back End

- java
- jsp/servlet
- oracle

### 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 myFavourite 인 노드들의 폰트를 bold처리하기.

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

## Front End

- **html**
- **css**
- JavaScript
  - [JavaScript Core](#)
  - [DOM](#)
  - [BOM](#)
- jQuery

## Back End

- java
- jsp/servlet
- oracle

### 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 부모노드의 li노드의 border 설정

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

## Front End

- html
- css
- JavaScript
  - [JavaScript Core](#)
  - [DOM](#)
  - [BOM](#)
- jQuery

## Back End

- java
- jsp/servlet
- oracle

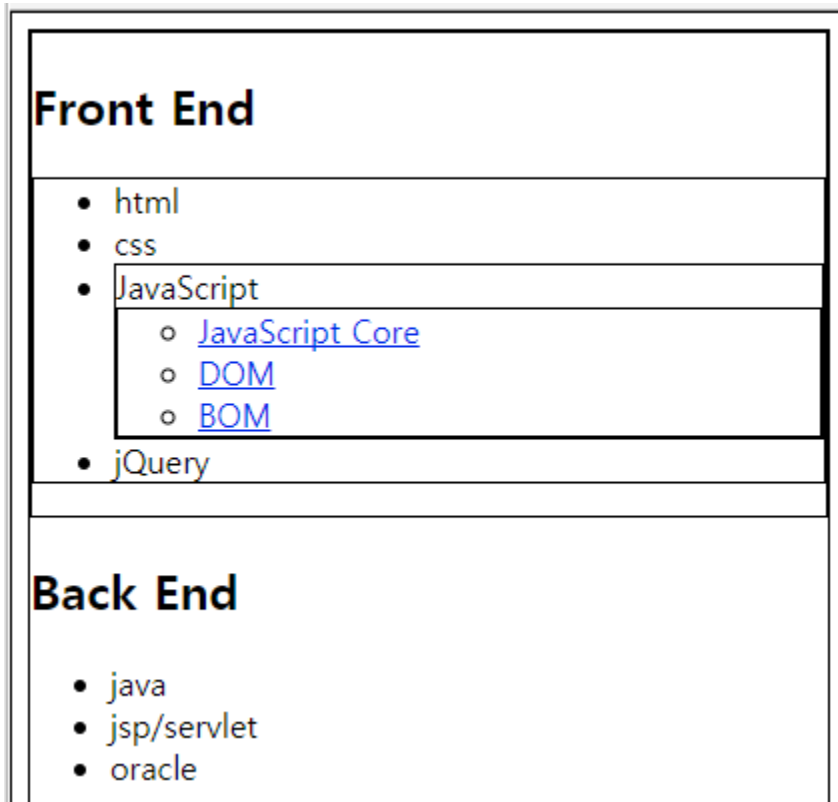
### 조상 노드 – parents()

`$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>
```

## Front End

- html
- css
- JavaScript
  - [JavaScript Core](#)
  - [DOM](#)
  - [BOM](#)
- jQuery

## Back End

- java
- jsp/servlet
- oracle

### 모든 이전 형제 노드 찾기 - 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>
```

## Front End

- html
- css
- JavaScript
  - [JavaScript Core](#)
  - [DOM](#)
  - [BOM](#)
- jQuery

## Back End

- java
- jsp/servlet
- oracle

### 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>
```

## Front End

- html
- css
- JavaScript
  - [JavaScript Core](#)
  - [DOM](#)
  - [BOM](#)
- jQuery

## Back End

- java
- jsp/servlet
- oracle

모든 다음 형제 노드 찾기 – **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>
```

## Front End

- html
- css
- JavaScript
  - [JavaScript Core](#)
  - [DOM](#)
  - [BOM](#)
- jQuery

## Back End

- java
- jsp/servlet
- oracle

## 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>
```

- 신메뉴
- menu1
- menu2
- menu3
- menu4
- menu5
- menu6

추가

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>
```

- menu1
- menu2
- menu3
- menu4
- menu5
- menu6
- 신메뉴

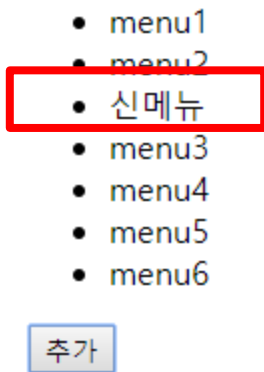
추가

### 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>
```

- menu1
- menu2
- menu3
- 신메뉴
- menu4
- menu5
- menu6

추가

## 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>
```

- menu3
- menu1
- menu2
- menu4
- menu5
- menu6

추가

### 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>
```

- menu1
- menu2
- menu4
- menu5
- menu6
- menu3

추가

## 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>
```

- menu1
- menu2
- menu4
- menu5
- menu3
- menu6

추가

### 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>
```

- menu1
- menu2
- menu4
- menu5
- menu3
- menu6

추가

## 노드 삭제

### 특정 노드 삭제

```
$대상.remove()
```

찾은 노드를 삭제한다

삭제버튼을 동적으로 추가하고, 클릭시 menu클래스의 자식노드li중 마지막 태그를 삭제해보자.

```
<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>
```

- menu1
- menu2
- menu3
- menu4
- menu5

추가

삭제

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>
```

추가

삭제

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>
```

## 마크업태그를 추가해보자.

html()

text()

<h2>마크업태그를 추가해보자.</h2>

html()

text()

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>
```

This page says:

1px solid rgb(0, 0, 0)

OK

div 노드의 width, height 스타일 알아내기

This page says:

width : 300px, height : 200px

OK

```
<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 때 사용한 `css()` 함수를 이용해 스타일을 지정할 수도 있다  
첫 번째 매개변수로 속성이름을 두 번째 매개변수로 속성 값을 지정한다

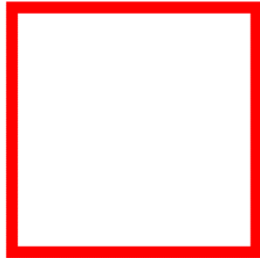
#test 노드의 border 스타일 지정

```
<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>
```

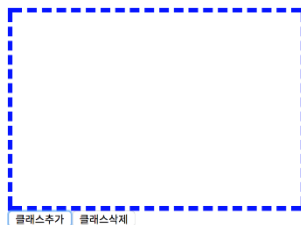
```
$대상.addClass("클래스이름1 클래스이름2 ...")
```

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>
```

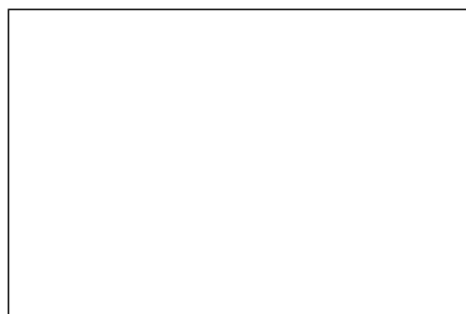


## 클래스 삭제

```
$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>
```



클래스추가 클래스삭제

## 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>
   <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 6. WEB API 활용**

## 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 콘텐츠가 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 사용법에 대하여 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
    // document.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 데이터를 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.addTo = 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.addTo 메서드를 호출할 함수를 만들자.

```
function addTo() {  
    var todo = document.getElementById('todo');  
    html5rocks.webdb.addTo(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 끌어다 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 핸들러 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 이벤트 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. 포맷을 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 <div> 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
- 이미지 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, Ericsson 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();
            }
        }
    }
}

```

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

```

open!!
message::new connection: 0:0:0:0:0:0:1
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 콘텐츠를 만들어 제공할 수 있음을 뜻한다. 화려한 그래픽 기반의 게임이나, 다양한 종류의 그래프, 이미지 합성 또는 변형, 드로잉 애플리케이션 등 마치 플래시로 생성된 것과 같은 콘텐츠를 제공할 수 있게 되는 것이다.

#### 간단한 예제

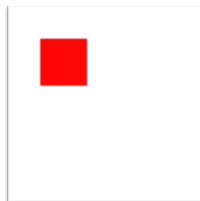
<canvas>요소에 빨간색 사각형을 그려보자. 아래 코드는 HTML 문서에 <canvas>요소를 작성한 것이다.

```
<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 벡터 그래픽(Scalable Vector Graphics)의 줄임말이다.

2 차원 벡터 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/example.svg" width="300" height="100" type="image/svg+xml"
codebase="http://www.adobe.com/svg/viewer/install/" />
<!-- embed 요소 사용 -->
<embed src="/svg/example.svg" width="500" height="200" type="image/svg+xml"
pluginpage="http://www.adobe.com/svg/viewer/install/" />
<!-- iframe 요소 사용 -->
<iframe src="/svg/example.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>











**KH정보교육원**