

WEB Front-End

목차

Chapter 1. HTML - 5

- 1.1 HTML이란
- 1.2 DTD, Document Type Definition
- 1.3 HTML 문서 구조
- 1.4 영역 관련 태그
- 1.5 글자 입력 관련 태그
- 1.6 목록 태그
- 1.7 하이퍼링크 태그 - <a>
- 1.8 테이블 태그 - <table>
- 1.9 폼 태그 - <form>
- 1.10 이미지 태그 -
- 1.11 비디오 태그 - <video>
- 1.12 오디오 태그 - <audio>
- 1.13 프레임 태그 - <iframe>
- 1.14 HTML5 시맨틱 태그

Chapter 2. CSS - 38

- 2.1 CSS 란
- 2.2 CSS를 HTML문서에 적용하는 방법

2.3 색상과 단위

2.4 선택자

2.5 CSS 주석

2.6 텍스트

2.7 레이아웃

2.8 테두리

Chapter 3. JavaScript - 68

3.1 자바스크립트 개요

3.2 자바스크립트 데이터타입

3.3 변수

3.4 주석

3.5 형변환

3.6 제어문

3.7 함수

3.8 변수와 함수의 관계

3.9 함수의 분류

3.10 자바스크립트 라이브러리

3.11 문서 객체 모델(DOM)

Chapter 4. jQuery - 123

4.1 jQuery 개요

4.2 jQuery 기능

4.3 jQuery 개발 환경 설정

4.4 노드 찾기

4.5 노드 생성/추가/삭제/이동

4.6 스타일 다루기

4.7 속성

Chapter1. HTML

1 HTML이란

웹의 최소 단위인 웹 페이지를 만드는 언어

HyperText Markup Language의 약자로 웹 페이지의 구조를 표현한다

태그로 되어있는 HTML요소 형태로 웹 페이지 구조를 표현한다

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 해준다

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

가장 많이 사용되는 HTML버전은 HTML4.01과 XHTML1.0, HTML5이며 각각의 DTD를 다르게 설정한다

2.1 HTML4.01 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 HTML문서와의 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 HTML5임을 브라우저에 알림

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

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

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

<title> : 제목표시줄에 나타날 내용

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

<h1> : body안에서 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 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 태그는 줄 바꿈이 적용되지 않아 옆으로 영역이 붙는다

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

4.1 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 , , <a>, 등의 태그가 있다

블록 요소로는 <div>, <p>, , , <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

HTML문서에서 글자 입력과 글자의 속성, 문단 설정, 정렬, 목록과 관련된 태그

5.1 제목 태그 - <h>

제목으로 사용될 문장을 표현하는 태그

<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>,

한 문단을 나타내는 <p> 태그

줄바꿈을 표현하는
 태그

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

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

- 줄 바꿈이 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 입력내용을 그대로 보여주는 태그 - <pre>

<p> 태그와 유사하지만 HTML문서 작성 시 입력한 내용을 그대로 보여준다

줄 바꿈, 정렬 상태 등을 입력한 그대로 보여준다

입력한 내용을 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>
```

가나다라마바사
아자차카타파하

5.4 글자 서식 태그

글자의 서식을 결정하는 태그

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 -

입력한 글자의 색이나 크기 또는 글꼴을 변경해주는 태그

 태그의 속성을 입력하여 글자의 속성을 원하는 대로 변경 가능

단, HTML5에서는 font 태그를 지원하지 않는다(CSS를 통해 글자의 속성을 변경한다)

5.5.1 글자 크기 설정 속성 - size

1(작음)부터 7(큼)까지 설정 가능하며 기본값은 3

```
<!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

5.5.2 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

5.5.3 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 태그

과 이 있으며 ul은 순서가 없는 목록을 나타내며, 은 순서가 있는 목록을 나타낸다

목록의 항목은 태그를 이용하여 나타낸다

항목은 자동으로 들여쓰기된다

- : 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

- : 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

7 하이퍼링크 태그 - <a>

클릭 시 다른 페이지로 이동시키는 하이퍼링크를 적용시켜주는 태그

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

글자나 문단 뿐만 아니라 그림 또는 html태그 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

target속성을 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 요소를 이용한다

8 테이블 태그 - <table>

표를 생성하는 태그

테이블의 셀 하나하나가 모두 태그로 이루어지며 복잡한 구조를 가지게 된다

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 표현

- 서울의 산을 테이블로 표현

```
<!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 확장 - rowspan, colspan 속성

테이블은 기본적으로 행과 열을 수를 모두 동일하게 갖는다

행이나 열의 수가 맞지 않으면 제대로 된 테이블의 모습을 갖추지 못 한다

우선, 테이블이 생성되는 구조를 확인하기 위해 HTML문서의 <head> 태그 안에 다음과 같이 적는다. CSS를 이용한 <style> 태그이다.

```

<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>
```

- 행과 열의 수가 맞지 않는 테이블

```
<!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>  
  </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

```
<!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 표현

```

<!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 태그

<form> 태그로 정보를 보낼 부분을 감싸주어야 한다

크게 action, method 두 가지 속성을 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 받을 부분을 지정하는 태그

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

type 속성은 정보를 어떤 형식으로 받을 지를 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

9.2.1 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>
```

9.2.2 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>
```

비밀번호 입력

.....

9.2.3 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>

```

9.2.4 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

9.2.5 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>

```



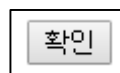
submit과 reset 속성값은 특수한 기능을 가지는데 submit 버튼은 <form> 태그의 action 속성값인 서버주소로 <form> 태그 내의 정보를 전달하는 기능을 한다. reset 속성의 버튼은 <form>태그 내에 입력된 정보들을 초기화시키는 기능을 가지고 있다.

<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>

```

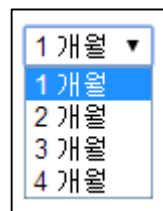
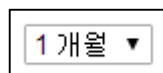


9.3 <select> - <option> 태그

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 -

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 풍선 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

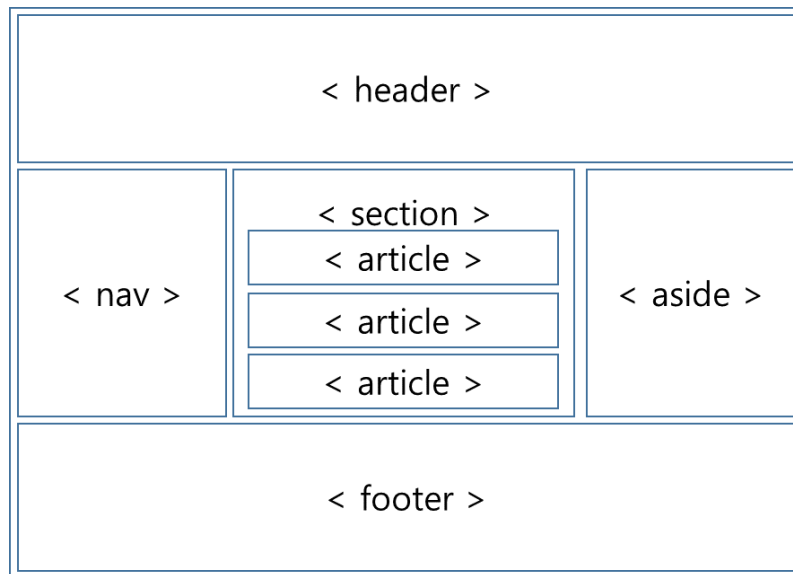
14 HTML5 시맨틱 태그

문서 구조를 정의하는 HTML5의 태그

<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 시멘틱 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

Chapter2. CSS

1 CSS 란

Cascading Style Sheets의 약자로 마크업 언어가 실제 표시되는 방법을 기술한 언어
HTML, XHTML, XML 같은 마크업 언어 문서의 스타일을 꾸밀 때 사용하는 스타일 시트

2 CSS를 HTML문서에 적용하는 방법

2.1 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>
```

2.2 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>
```

2.3 css파일을 로드해 적용하는 방식

별도로 CSS파일을 작성해 HTML문서에 로드하여 스타일을 적용하는 방법
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 용이해진다

2.3.1 @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;
}
```

2.3.2 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>

```

3 색상과 단위

3.1 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) |

3.1.1 RGB 방식

색을 표현할 때 색상 이름을 사용하는 방식

black, white, red, green, tomato 등이 있다

3.1.2 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

3.1.3 RGBA 방식

RGB방식에 투명도가 포함된 방식

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

3.1.4 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

3.1.5 HSLA 방식

HSL방식에 투명도가 포함된 방식

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

3.2 웹 안전 색상

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

3.3 크기 단위

3.3.1 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 : 파이크. 1pc = 1/6in
- cm : 논리 센티미터. 1in = 2.54cm
- mm : 논리 밀리미터. 1in = 25.4mm

3.3.2 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

4 선택자

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

4.1 선택자 종류

4.1.1 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>
```

4.1.2 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>

```

4.1.3 클래스 선택자

| 선택자 패턴 | 의미 |
|----------|------------------------------|
| .myClass | 클래스 속성값이 myClass로 지정된 요소를 선택 |

특정 class 속성값을 가지는 요소를 선택

선택하려는 요소의 클래스 속성값 앞에 . (마침표)를 추가해서 작성한다. 해당 클래스 속성을 가지는 모든 요소를 선택한다.

클래스 선택자 앞에 태그를 붙여주면 해당 태그 요소의 범위 내에서 해당 클래스를 선택

- 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>

```

4.1.4 아이디 선택자

| 선택자 패턴 | 의미 |
|--------|---------------------------|
| #myId | 아이디 속성값이 myId로 지정된 요소를 선택 |

특정 id 속성값을 가지는 요소를 선택

#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

```
<html>
<head>
<style>
/* CSS */
#id1 { background: yellow; color: green; }
div#id2 { background: red; color: blue; }
</style>
</head>
<body>
<p id="id1">아이디 선택자</p>

<p id="id2">아이디 선택자 미 적용</p>
<div id="id2">아이디 선택자 적용</div >
</body>
</html>
```

4.1.5 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

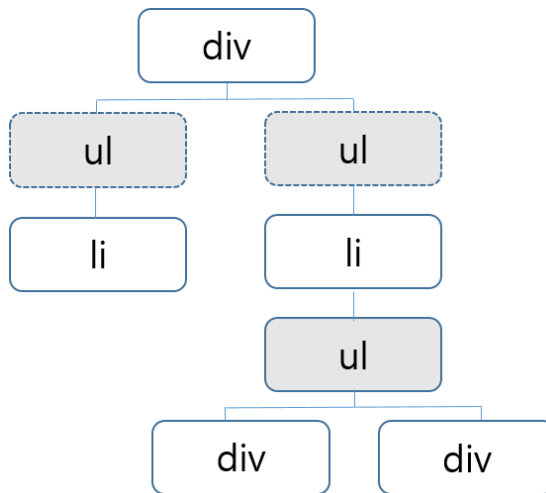
요소 사이에 > 기호를 사용하는 자식 선택자는 부모 요소의 바로 아래 자식 요소만 선택한다

- 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>

```

4.1.6 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

4.1.7 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

4.1.8 부정 선택자

선택자 패턴	의미
E.not(S)	S가 아닌 E 요소 선택

조건을 주어 만족하지 않는 요소를 선택한다

- 부정 선택자 예제

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

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

4.2 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.

4.2.1 !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.

5 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 */
```

6 텍스트

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

6.1 font-size

텍스트의 크기를 설정하는 속성

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

```
</html>
```

6.2 color

텍스트의 색상을 지정

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

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

6.3 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>
```

6.4 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>

```

6.5 font-variant

대소문자에 대한 스타일

소문자를 작은 대문자 형태로 표현하도록 설정할 수 있다

```

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

```

6.6 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>
```

6.7 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, helvetica, sans-serif;

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

6.8 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 를 가운데 정렬시키기 위해 블록 요소인 <div>를 이용

6.9 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>

```

6.10 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>
```

7 레이아웃

7.1 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

7.1.1 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

7.2 CSS 여백속성 - margin, padding



7.2.1 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>
```

7.2.2 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>
```

7.3 CSS 흐름속성 - float, clear

7.3.1 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

7.3.2 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)

7.4 overflow와 Text-overflow 속성

7.4.1 overflow

컨텐츠가 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

7.4.2 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

7.5 display, visibility 속성

7.5.1 display

요소를 표시하는 방법을 지정

요소의 블록 모델을 바꿀 수 있는 속성이다

7.5.2 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

7.6 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

7.6.1 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.

7.6.2 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

7.6.3 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 들여놓거나 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: auto를 지정하면 부모 요소의 왼쪽에 정렬되고 margin-left: auto를 지정하면 부모 요소의 오른쪽에 정렬된다. margin-left: auto와 margin-right: auto를 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.

7.6.4 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

7.6.5 절대 위치(absolute) 박스

원래 위치를 벗어나 위/아래에 레이어를 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

7.6.6 플로트(float) 박스

플로트 속성이 지정된 요소는 원래 위치를 벗어나서 인접한 블록 요소의 테두리와 배경 위를 덮는다. 플로로 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.

8 테두리

8.1 border

테두리를 설정하는 속성

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

몇몇 속성을 생략해도 상관없다

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

8.1.1 border-width

테두리의 두께를 지정한다

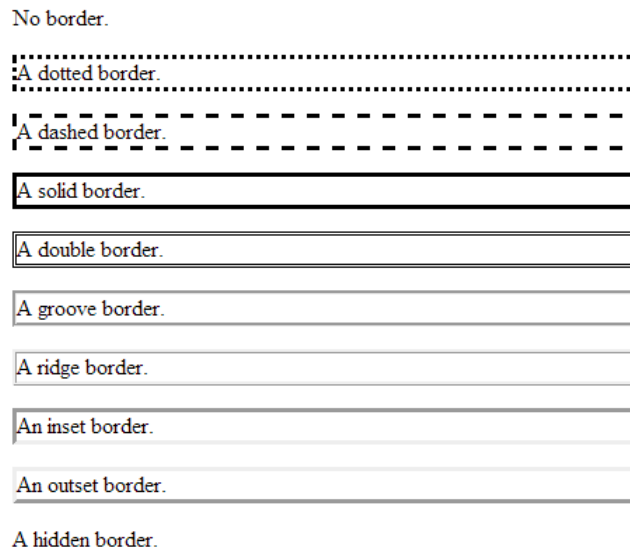
px단위로 많이 지정하며, thin, thick 과 같은 단어로 설정 가능하다

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

8.1.2 border-style

테두리 스타일을 지정한다

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



8.1.3 border-color

테두리 색상을 지정한다

8.2 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>
```

8.3 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>
```

Chapter3. JavaScript

1 자바스크립트 개요

선마이크로시스템즈와 네스케이프에서 공동으로 개발한 스크립트 언어

HTML문서의 동작을 정적으로 표현할 수 있는 언어이다. HTML Form값을 가공하거나 검사하여 서버쪽으로 보낸다든지 사용자의 입력을 제어한다든지 사용자에게 메시지를 보내는 등으로 활용할 수 있다.

HTML 문서 안에 자바스크립트로 작성된 코드를 넣어 둠으로써 브라우저에서 실행가능하도록 한다. 인터프리터 언어로써 브라우저에 의해 실행될 때마다 번역이 이루어진다. 자바스크립트는 스크립트 언어로써 스스로 실행 가능하지 않다. 따라서 자바스크립트를 지원하지 않는 브라우저에서는 자바스크립트 코드를 무시하게 된다.

1.1 자바스크립트 구조

HTML 언어에 삽입하여 사용

<head> 태그나 <body>태그 안에 <script>태그와 </script> 태그 사이에 자바스크립트 문서를 입력해서 내용을 정의한다. <head> 태그에는 문서가 로딩될 때 미리 실행할 내용을 기록하고, <body> 태그는 특정 부분에서 실행될 때 입력한다.

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

위 소스의 <!-- //--> 는 HTML문서의 주석코드를 자바스크립트 코드에 삽입한 것이며 자바스크립트 코드의 주석으로 처리되지 않는다. 구 버전의 브라우저나 몇몇 특정 브라우저에는 자바스크

립트 엔진이 없는 경우가 있는데 브라우저에 자바스크립트 엔진이 없을 경우 자바스크립트 코드를 제대로 인식하지 못하는 경우가 있다. 이 경우 자바스크립트 코드로 인식하지 못 하고 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 필요 없으며 HTML코드 내에 삽입되어 소스가 공개되어 있다.

2 자바스크립트 데이터타입

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

2.1 문자열(String)

문자열을 표현하는데 사용하는 데이터 타입

16비트 유니코드 문자의 연결구조이다

객체로서 활용가능하다(property와 method를 가진다)

2.1.1 length property

문자열의 길이를 알 수 있는 String의 프로퍼티

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

> 5
```

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

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

String타입에서 자주 사용되는 메소드

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

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

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

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

2.2 숫자(Number)

숫자를 표현하거나 산술 연산에 사용되는 데이터 타입

Math라는 내장객체를 이용하여 수학함수의 결과를 숫자타입으로 얻을 수도 있다

2.2.1 Number 캐스팅(Casting) - parseInt(), parseFloat()

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

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

<script>
```

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

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

```
<script>
console.log(parseInt("010"));          // 진법을 지정하지 않아 8진수로 인식
</script>
>8
```

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

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

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

2.2.2 Not a Number

문자열을 숫자로 데이터 형 변환 시에 문자열 대상이 숫자를 표현하고 있지 않을 경우 Not a Number의 약자인 NaN을 리턴하며 숫자의 형태가 아니라는 뜻이다.

```
<script>
console.log(parseInt("Hello", 10));
</script>
>NaN
```


2.2.3 isNaN()

NaN인지 검사하는 내장함수

```
<script>
console.log(isNaN(Nan));
</script>
>true
```

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

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

2.2.4 Infinity, -Infinity, isFinite()

양의 무한대의 숫자를 나타내는 Infinity, 음의 무한대의 숫자를 나타내는 -Infinity

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

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

무한대가 아닌 숫자인지 판별하는 내장함수 isFinite()

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

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

2.3 널(null)과 undefined

널(null)은 값이 없음을 나타낸다

undefined는 초기화 되지 않았거나 선언되지 않았거나 값이 할당되지 않았음을 나타낸다

null은 의도적으로 값을 비워둔 상태이며 undefined는 처음부터 어떠한 값도 할당되지 않은 것을 뜻한다

2.4 Boolean

논리값을 표현하는 데이터 타입

true와 false값을 가진다

true/false 값을 통해 표현식의 참과 거짓을 구분할 수 있다.

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

3 변수

변수를 선언하여 데이터를 저장할 수 있는 공간을 만들어 사용할 수 있다

3.1 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 선언과 초기화하기

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

</script>
```

3.2 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>
```

3.3 변수 선언 시 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 사용 될 변수는 모두 대문자로 선언하는 것이 좋다

3.4 변수 값 확인하기

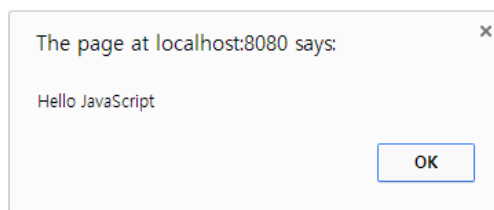
3.4.1 alert()

특정 정보를 메시지 창으로 사용자에게 알려주기 위해 사용
문법 테스트 용으로 사용할 수 있다

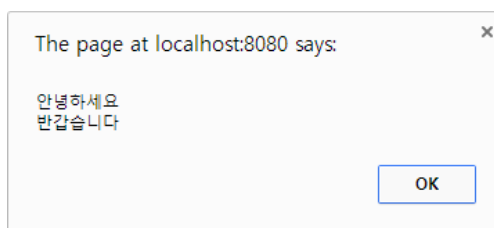
```
<script>  
alert(데이터);  
</script>
```

- 크롬브라우저에서 alert 확인

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



- 변수에 저장된 값 출력



3.4.2 document.write()

HTML문서의 <body> 영역에 내용을 출력한다

```
<script>
```

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

3.4.3 console.log()

브라우저의 디버깅 기능(개발자 도구)의 화면에 콘솔영역으로 출력 한다

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

4 주석

작성한 코드에 성명을 달아놓거나 코드가 실행되지 않도록 하는 소스
개발자 간의 협업이나 유지보수를 위해 사용한다
한 줄 주석과 여러 줄 주석이 있다

- 한 줄 주석

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

- 여러 줄 주석

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

4.1 배열

배열이란 한 개의 데이터만을 저장할 수 있는 변수와 달리 변수 하나에 여러 개의 데이터를 담
아 사용할 수 있는 것이다
서로 연관 있는 데이터를 하나의 배열에 담아 관리한다

4.1.1 배열 생성

변수를 선언하는 것과 동일하지만 여러 개의 데이터를 “[]”로 묶어 대입하여 사용한다

```
<script>
var 변수이름 = [데이터1, 데이터2, ... ];
</script>
```

4.1.2 배열 요소 접근

선언된 배열의 데이터에 접근하기 위해서는 인덱스를 “[]”안에 적어 접근한다

0번째 인덱스부터 사용할 수 있으며 배열이 가지고 있는 데이터의 개수를 넘어 인덱스를 지정할 수 없다

```
<script>
배열이름[n];
</script>
```

5 형변환

숫자를 문자로 변환하거나 문자를 숫자로 변환하는 것처럼 데이터 타입을 바꾸는 것
형변환에는 암시적 형변환과 명시적 형변환이 있다

5.1 암시적 형변환

자바스크립트 엔진이 필요에 의해 암시적으로 형을 자동 변환 시키는 것
상황에 따라 바뀌는 데이터 타입이 달라진다

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

5.2 암시적 형변환

개발자가 코드로 직접 어떤 형으로 바꿀지 명시하여 형을 변환 시키는 것
문자를 숫자로, 숫자를 문자로 변경하는 방법을 많이 사용한다

5.2.1 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

5.2.2 숫자를 문자로 형변환

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

6 제어문

프로그램의 흐름을 변화시킬 수 있는 요소

6.1 조건문

6.1.1 if 조건문

자바스크립트에서 가장 일반적으로 쓰이는 조건문

- if 조건문의 기본 형태

```
if ( 표현식 ){  
    // 수행 문장  
}
```

표현식의 결과가 true이면 중괄호 {}안의 문장을 실행

false이면 중괄호 코드 무시

- 기본적인 if 조건문

```
<script>  
if(100 > 200) {  
    // 100 > 200 이 참일 때 실행  
    alert('100 > 200 -> true');  
}  
  
alert('작업완료');  
</script>
```

6.1.2 if else 조건문

기본 if 조건문은 표현식이 참일 때만 수행하고 거짓일 때는 코드를 무시하게 된다

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

- if else 조건문의 기본 형태

```
if ( 표현식 ){  
    // 표현식이 참일 때 수행되는 문장  
}  
  
else {
```

```
// 표현식이 거짓일 때 수행되는 문장
}
```

표현식의 결과가 true이면 if 조건문에 붙어있는 중괄호 {}안의 문장을 실행
false이면 else 문에 붙어있는 중괄호 {}안의 문장 실행

- 기본적인 if else 조건문

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

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

alert('작업완료');
</script>
```

6.1.3 if else if 조건문

if 조건문의 표현식이 거짓일 경우 무조건 수행되는 else 문과 다르게 한번 더 표현식을 검사할 수 있도록 만든 조건문

- if else 조건문의 기본 형태

```
if ( 표현식 ) {
    // 표현식이 참일 때 수행되는 문장
```

```

}
else if ( 표현식 ) {
// 표현식이 거짓일 때 수행되는 문장
}
else if ( 표현식 ) {
// 표현식이 거짓일 때 수행되는 문장
}
else {
// 표현식이 거짓일 때 수행되는 문장
}

```

else if 문은 여러 번 올 수 있다

else 문은 연결된 if 문의 마지막에 한번만 올 수 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>

```

6.1.4 중첩 if 조건문

if문, else if문, else문 내에 조건문을 추가적으로 작성한 것이다
조건식을 더 상세하게 조절할 수 있는 장점이 있다

- 1-100 사이의 숫자인지 판별한 수 홀짝을 판별

```

<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>

```

6.1.5 switch 문

- switch 문의 기본 형태

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

case 문은 여러 번 올 수 있다

default 문은 마지막에 위치해야 하고 생략 가능하다

- 훌쩍 구분하기

```
<script>  
var num = 7;  
  
switch(num%2) {  
  case 0:  
    alert('짝수');  
    break;  
  case 1:  
    alert('홀수');  
    break;  
}  
</script>
```

6.1.6 삼항연산자를 이용한 간단한 조건문

삼항조건연산자를 이용해 조건문과 같은 기능을 수행할 수 있다

- 삼항연산자

표현식 ?참일 때 수행 :거짓일 때 수행

- 기본적인 삼항연산자 조건문

```
<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>
```

6.2 반복문

6.2.1 for 반복문

반복 횟수에 유의하며 반복문을 작성한다

- 기본적인 for 조건문

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

- 수행순서
 1. 초기식을 실행
 2. 조건식 비교. 거짓이면 반복문 종료
 3. 수행코드 실행
 4. 증감문 수행
 5. 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(arr.length) 만큼 반복하여 간단히 배열의 요소를 모두 출력할 수 있다

6.2.2 for in 반복문

배열이나 객체를 이용한 반복문을 쉽게 다룰 수 있도록 for in 반복문을 제공

- 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>
```

6.2.3 while 반복문

if 조건문과 비슷한 형태를 가지며 조건식이 참일 경우 한번 만 수행하는 if조건문과 다르게 while반복문은 조건식이 참이라면 지속적으로 문장을 반복한다
조건식에 유의하면서 반복문을 작성한다

- 기본적인 while 조건문

```
while( 조건식 ) {
    수행코드
}
```

조건식이 참이라면 수행코드를 반복

- while문을 이용한 무한 반복(실행 금지)

```
<script>
while(true) {
    alert(arr[i]);
}

</script>
```

조건식이 참에서 변경되지 않으면 수행코드는 무한히 반복하게 된다

while 반복문을 종료되도록 하려면 조건을 변화시킬 수 있는 요소가 있어야 한다

- while 반복문 내에서 조건을 변경

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

while 문이 반복될 때마다 i의 값이 1씩 증가되므로 무한루프에 빠지지 않게 된다

for문의 초기식, 조건식, 증감식을 생각하면서 while문을 작성하면 편하게 while문을 사용할 수 있다

6.2.4 do while 반복문

while 문과 비슷하지만 do문을 먼저 사용하고 마지막에 while(조건식) 문을 적어 반복을 제어한다

while문은 조건식을 먼저 판단하여 반복을 수행할지 결정하지만 do-while문은 우선 수행코드를 한번 수행한 후에 조건식을 판단하여 반복을 더 수행할지 결정한다

- 기본적인 do while 조건문

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

수행코드는 최소한 한번 실행된다

- do while문을 이용한 반복문

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

```
} while(i<5);  
</script>
```

6.2.5 중첩 반복문

조건문을 중첩시킨 것과 마찬가지로 반복문도 중첩시켜 사용할 수 있다

- 중첩 for문을 이용한 별탑 출력

```
<script>  
var output = "";  
  
for(var i=0; i<5; i++) {  
    for(var j=0; j<=i; j++) {  
        output += '*';  
    }  
    output += '\n';  
}  
  
alert(output);  
</script>
```

6.3 break문 & continue문

6.3.1 break문

switch문이나 반복문을 중단시킬 때 사용

switch문이나 반복문은 break문을 만나면 코드를 중단하고 벗어나게 된다

- 조건문과 break문을 섞어 무한 반복문 중단

```
<script>  
var i=0;  
while(true) {  
    alert(i + '번째 반복 수행');
```

```

// i가 10과 같다면 while 반복문 중단
if(i == 10) {
    break;
}

i++;
}
</script>

```

- 반복 수행할 때마다 확인하며 진행

```

<script>
// 조건식이 true로 무한 반복
for(var i=0; true; i++) {
    alert(i + '번째 반복');

    // 반복을 진행할지 확인
    if(!confirm('계속 수행하시겠습니까?')) {
        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. 함수의 호출과 반환에 대한 자세한 것은 다음 장에서 설명한다.

6.3.2 continue문

반복문에서 사용 가능

반복문에서 수행하던 코드를 중단하고 다음 반복을 진행을 지시하는 코드이다

while문은 조건식을 검사하게 되며, for문은 증감식을 수행한다.

- continue문 사용

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

alert('작업 완료');
</script>
```

7 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.

7.1 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 키워드를 먼저 적고 함수 이름을 옆에 적어야 한다. 함수의

이름은 유일해야 하고 만들려는 함수의 기능을 함축한 의미를 가지도록 하는 것이 좋다. 함수의 기능을 {} 안에 구현한다.

정의된 함수를 호출하기 위해서는 "함수이름()" 형식으로 사용한다

7.2 함수 형태 2 – 매개변수가 있는 함수

기본 함수 형태에서 매개변수가 추가된 형태

```
<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 넣어주어야 한다. 호출방식은 "함수이름(전달값1, 전달값2, ...)" 형식으로 사용한다

7.3 함수 형태 3 – 리턴값이 있는 함수

리본 함수 형태에서 리턴값이 추가된 형태

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

```
</script>
```

리턴값은 함수가 종료되는 시점에 함수를 호출한 함수 외부에 정보를 전달하는 역할을 한다. 매개변수는 함수가 시작되며 값을 전달받는 역할을 한다면 리턴값은 함수가 종료되며 값을 전달해주는 역할은 한다.

또한 함수는 return 코드를 수행하면 함수의 작업을 종료하게 된다. 이를 이용하여 함수를 종료시키고 싶은 시점이 있다면 리턴값이 없더라도 return 구문을 이용해 함수를 종료시킬 수 있다.

7.4 함수 이름 주의 사항

함수 이름을 지을 때 변수명과 마찬가지로 주의 사항이 존재한다

1. 숫자로 시작하면 안 된다
2. 대소문자를 구분한다
3. 키워드를 함수명으로 사용할 수 없다
4. 함수명은 함수의 기능을 대표할 수 있는 단어를 사용하는 것이 좋다
5. 일반적으로 함수명은 소문자로 시작하는 동사형식의 단어를 사용한다

8 변수와 함수의 관계

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

함수를 변수에 담으면 변수명은 함수명과 동일하게 동작한다

```
<script>
function hello() {
    alert("Hello JavaScript");
}
hello();
```

```
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

9 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 만들어져 자바스크립트에서 제공하는 함수를 말한다.

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

9.1 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.

9.2 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>
```

10 자바스크립트 라이브러리

자바스크립트가 개발자를 위해 기본적으로 제공해주는 기능

10.1 타이머 함수

일정 시간마다 특정 구문을 실행하고자 할 때 사용하는 기능

10.1.1 주요 기능

타이머 함수는 모두 전역 객체인 window에 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

10.1.2 일정 시간마다 주기적인 실행 – setInterval()

```
<script>
var timerID = setInterval(func, duration);
</script>
```

func : 지연 시간마다 타이머 함수에 의해 호출되는 함수

duration : 지연 시간, 단위는 밀리초

- 1000밀리초(1초) 마다 알림창을 띄우는 예제

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

- hello() 함수를 익명 함수로 작성한 예제

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

10.1.3 일정 시간 후 한번만 실행 – setTimeout()

```
<script>
var timerID = setTimeout(func, duration);
</script>
```

func : 지연 시간이 지난 후 타이머 함수에 의해 호출되는 함수

duration : 지연 시간, 단위는 밀리초

- 2000밀리초(1초) 후 알림창을 띄우는 예제

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

10.1.4 타이머 멈추기 – clearInterval(timerID);

```
<script>
clearInterval(timerID);
</script>
```

timerID : 제거할 타이머 ID

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

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

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

10.2 Math 클래스

숫자와 관련된 유용한 기능이 담겨있는 클래스

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

10.2.1 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

Math클래스의 기능은 인스턴스 생성 없이 즉시 사용할 수 있다

Math.abs(-10) 과 같이 사용하면 된다

10.2.2 랜덤 숫자 만들기 - Math.random()

random() 함수는 0과 1 사이의 실수 형태의 숫자를 반환한다

```
<script>
var value = Math.random() * 원하는 숫자

</script>
```

- 1초마다 50~100 사이의 정수 출력하는 예제

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

</script>
```

10.2.3 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>
```

10.2.4 내림 값, 올림 값 구하기 – Math.floor(), Math.ceil()

Math.floor() 함수는 입력된 실수의 소수점 부분을 버림한 값을 반환

Math.ceil() 함수는 입력된 실수의 소수점 부분을 올림한 값을 반환

```
<script>
document.write(Math.floor(15.678));

</script>
```

15

<pre><script> document.write(Math.ceil(15.678)); </script></pre>
16

10.3 String 클래스

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

10.3.1 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
substr(start)	start 번째부터 문자열 추출
substr(start, count)	start 번째부터 count개수만큼 문자열 추출
toLowerCase()	모든 문자열을 소문자로 변환
toUpperCase()	모든 문자열을 대문자로 변환

trim()	좌우 공백 제거
--------	----------

10.3.2 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>
```

10.3.3 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>
```

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

10.4 Date 클래스

날짜 및 시간과 관련된 기능이 담겨있는 클래스

10.4.1 주요 기능

- 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
getFullYear()	로컬시간을 사용하여 년도를 반환. getFullYear()를 사용하는 것이 좋다

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

```
<script>
var hours = Date인스턴스.getHours();
</script>
```

0-23의 정수로 시간을 반환한다

```
<script>
var minutes = Date인스턴스.getMinutes();
</script>
```

0-59의 정수로 분을 반환한다

```
<script>
var seconds = Date인스턴스.getSeconds();
</script>
```

0-59의 정수로 초를 반환한다

```
<script>
var mSeconds = Date인스턴스.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>
```

```
</script>
```

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

```
<script>  
var year = Date인스턴스.getFullYear();  
</script>
```

네 자리 숫자의 년도 반환

```
<script>  
var month = Date인스턴스.getMonth();  
</script>
```

0-11의 정수로 월 반환

```
<script>  
var date = Date인스턴스.getDate();  
</script>
```

1-31의 정수로 날짜 반환

```
<script>  
var day = Date인스턴스.getDay();  
</script>
```

0(일)-6(토)의 정수로 요일 반환

- 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

10.5 Array 클래스

배열을 리터럴 방식([데이터1, 데이터2, ...])으로 사용하더라도 String과 마찬가지로 인스턴스를 자동으로 생성해 변수에 담게 되어 있다

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

10.5.1 주요 기능

- 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

10.5.2 배열 생성하기

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

- 리터럴 방식

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

- Array 객체 생성 방식

<pre><script></pre>

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

10.5.3 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>
```

10.5.4 특정 위치 배열 요소 접근

배열변수[index]

배열의 N번째 요소에 접근하려면 배열변수의 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>
```

10.5.5 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>
```

10.5.6 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>
```

11 문서 객체 모델(DOM)

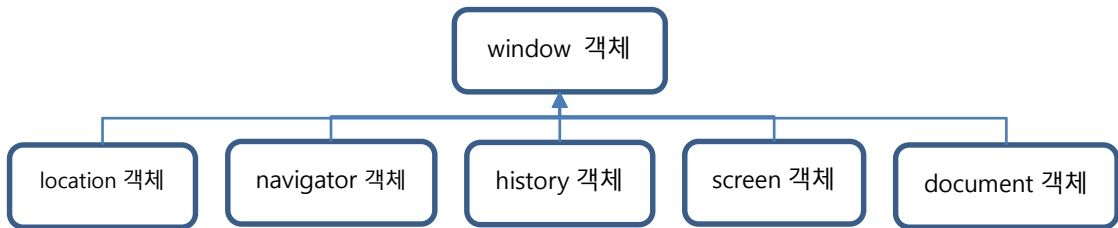
HTML 문서의 모든 요소에 접근하는 방법을 정의한 자바스크립트 API

자바스크립트 코드에서 동적인 HTML을 만들기 위해 DOM 객체에 접근하여 조작할 수 있다

웹 화면에 보이는 요소를 조작하기 위한 기능을 모아 놓은 라이브러리 덩어리(API)로써 DOM(Document Object Model), 자바스크립트 DOM으로 부른다

11.1 브라우저 객체 모델(DOM)

- 대표적인 브라우저 관련 객체



이 중 document 객체는 별도로 DOM(Document Object Model) 이라고 부른다.

11.2 DOM 구조

DOM은 정의 부분과 구현 부분으로 나뉘어져 있으며 정의부분(명세)에는 실제 동작하는 구현 소스 없이 웹 페이지 문서를 조작할 때 지켜야하는 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 해놓은 브라우저일 경우 웹 표준을 지원하지 않는 브라우저라고 부른다.

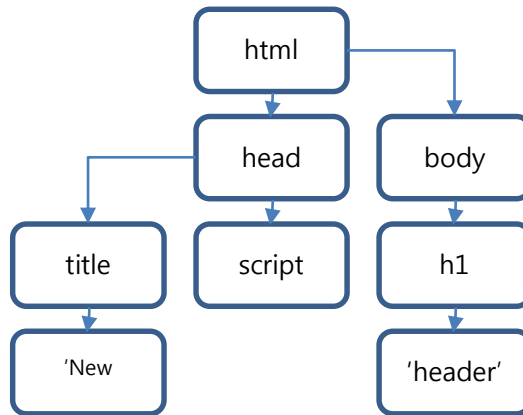
- 간단한 HTML 문서

```

<!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



11.3 DOM과 HTML 페이지

HTML문서를 이용하여 웹 페이지를 작성하여 브라우저에서 읽어들이면 브라우저는 HTML 문서의 태그 정보를 파싱하여 DOM 객체로 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 칭한다.

11.4 주요 DOM 객체

11.4.1 Node 객체

노드를 다루는 기본 기능과 프로퍼티를 제공

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.

11.4.2 Element 객체

주석 노드와 텍스트 노드를 제외한 나머지 노드를 통합해서 칭하는 용어

태그 요소의 기본 기능과 프로퍼티를 제공하며 속성을 제거하거나 속성값을 구하고, 설정할 수

있는 기능을 제공한다. 이벤트와 관련된 기능을 사용할 수 있다.

11.4.3 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, 태그의 DOM 객체인 HTMLImageElemnt, <body> 태그의 DOM 객체인 HTMLBodyElement와 같은 객체의 부모 객체이다.

11.4.4 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, tagName등으로 특정 노드를 찾는 기능, 이벤트를 등록시키는 기능까지 제공하는 객체

11.4.5 HTMLDocument 객체

HTML 문서 전용 Document 객체

HTML 페이지 로딩 후 파싱 단계에서 만들어진 html, head, body 객체를 비롯하여, 페이지에 작성된 태그와 매칭되는 모든 Node 객체를 가지고 있는 객체이다

11.5 문서 객체 만들기

- DOM의 노드 생성 메소드

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

- h1요소와 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 추가 시켜주어야 한다.

- 노드 연결 메소드

메소드	설명
appendChild(node)	객체에 노드를 연결

- h1요소와 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>

```



11.5.1 속성 지정하기

- 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>
```



11.5.2 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>
```

11.6 문서 객체 가져오기

이미 존재하는 HTML 태그를 자바스크립트로 가져오는 방법

11.6.1 ID로 가져오기

- 노드 추출 메소드

메소드	설명
getElementById(id)	ID속성이 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>
```

JavaScript with DOM

11.6.2 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>

```

11.7 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

Chapter4. jQuery

1 jQuery 개요

자바스크립트의 DOM(Document Object Model)을 이용해 HTML문서의 노드, 스타일, 속성, 이벤트, 위치 및 크기 등을 다룰 수 있다. DOM 객체는 HTML 문서의 모든 요소에 접근하는 방법을 정의한 API이다

jQuery는 자바스크립트의 DOM 객체를 이용한 작업을 좀 더 쉽게 만들어주는 자바스크립트 라이브러리이다. 자바스크립트로 작업하는 것보다 jQuery를 이용하면 효율적으로 코드를 작성할 수 있다. 또한, CSS에서 사용하는 선택자를 이용하여 jQuery에서 노드를 쉽게 찾을 수 있다는 장점이 있다.

- jQuery 라이브러리 사이트

<http://jquery.com>

2 jQuery 기능

2.1 jQuery DOM

jQuery의 핵심기능

노드 찾기, 수정하기, 삭제하기 등 노드와 관련된 모든 기능을 제공한다

2.2 jQuery Ajax

서버와 데이터를 쉽게 주고 받을 수 있는 통신 기능을 제공하며 서버로부터 받은 데이터를 jQuery DOM과 연계할 수 있어 동적인 페이지를 쉽게 제작할 수 있게 한다

2.3 jQuery 플러그인

jQuery의 기능을 확장하여 사용할 수 있도록 만들어진 기능들

다양한 기능들의 플러그인이 존재하며 필요한 기능이 있다면 미리 만들어져있는 플러그인을 찾아서 활용할 수 있다

2.4 jQuery 효과

웹 페이지를 구성할 때 사용하는 효과 기능을 기본적으로 포함하고 있다

DOM 요소와 연계하여 효과를 쉽게 적용할 수 있다

3 jQuery 개발 환경 설정

3.1 jQuery 라이브러리 삽입

3.1.1 CDN 이용

jQuery의 다양한 CDN을 이용해 웹페이지에 삽입

- 구글 Ajax API CDN : <http://ajax.googleapis.com/ajax/libs/jquery/1.11.0/jquery.min.js>
- 마이크로소프트 CDN : <http://ajax.aspnetcdn.com/ajax/jquery/jquery-1.11.0.min.js>
- jQuery CDN : <http://code.jquery.com/jquery-1.11.0.min.js>

- jQuery CDN을 이용하여 웹페이지에 jQuery 라이브러리 삽입 예

```
<script type="text/javascript" src="http://code.jquery.com/jquery-1.11.0.min.js"></script>
```

3.1.2 다운로드한 파일을 이용

jQuery 홈페이지를 통해 jQuery 파일을 다운받아 링크를 웹페이지에 삽입

- jquery-1.11.0.min.js 파일을 다운받아 라이브러리 삽입하는 예제

```
<script type="text/javascript" src="jquery-1.11.0.min.js"></script>
```

3.1.3 jQuery 버전

jQuery 사이트를 보면 2.0x 버전과 1.10x 버전 두 가지가 존재한다

2.0x 버전은 표준을 지원하는 브라우저의 기능만으로 만들어진 라이브러리이며 1.10x 버전은 표준 및 비표준 모두 지원한다. 추후에는 1.10x 버전이 사라지고 2.0x 버전만 남겠지만 현재에는 아직 비표준 브라우저(ex. IE7)를 사용하기도 하므로 1.10x 버전의 사용을 고려해야 한다.

3.2 ready() 메소드 설정

Java 코드의 프로그램 시작점인 main() 메소드처럼 jQuery의 진입점인 ready() 메소드가 존재

- jQuery 진입점 ready() 메소드 설정 4가지 방법

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

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

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

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

어떠한 방법을 써도 상관없지만 3번째 방식을 공식적으로 추천하고 있다.

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 코어대신 사용하는 예제

```
<script type="text/javascript">
```

```
var J = jQuery;

J(document).ready(function() {

    alert("안녕하세요");

});

</script>
```

3.3 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-1.11.0.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

4.1 일반 노드 찾기

4.1.1 아이디 이름으로 노드 찾기

```
$("#아이디 이름")
```

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

- 아이디 선택터 예제

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

4.1.2 클래스 이름으로 노드 찾기

```
$(".아이디 이름")
```

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

- 클래스 선택터 예제

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

4.1.3 태그 이름으로 노드 찾기

```
$("태그 이름")
```

태그 이름을 매개변수로 \$()함수를 호출

- 태그 셀렉터 예제

```
<script>
$(document).ready(function() {
    $("p").css("border", "3px solid #f00");
});
</script>
```

4.1.4 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

4.2 찾은 노드 다루기

\$() 함수를 통해 노드를 찾으면 찾은 노드에 해당하는 객체를 반환한다

찾은 노드를 변수에 담아 활용하거나 반환된 객체를 이용해 노드를 다룰 수 있다

- div 태그들을 찾아 \$divs 변수에 담기

```
<script>
var $divs = $("div");
</script>
```

4.2.1 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("length = " + $("div").length);
});
</script>
```

4.2.2 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 $list = $("ul li")
    var $list_1 = $list.eq(1);

    $list_1.css("border", "1px solid #f00");
});
</script>
```

```
<script>
$(document).ready(function() {
    // 한 줄로 표현 가능
    $("ul li").eq(1).css("border", "1px solid #f00");
});
</script>
```

4.2.3 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 #f00";
});
</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 #f00";
});
</script>
```

4.2.4 순차적으로 접근하기 – each() 메소드

each() 메소드를 사용하여 찾은 노드를 순차적으로 접근할 수 있다

Java의 foreach와 비슷한 동작을 한다

```
<script>
$(document).ready(function() {
    var $list = $("ul li");
    $list.each(function(index) {
        console.log(index); // 찾은 노드들의 개수만큼 alert() 호출
    });
});
</script>
```

```
});  
});  
</script>
```

4.2.5 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() {  
    $("#content").find(".redBox").css("border", "2px solid #f00");  
});  
</script>
```

4.3 자식 노드 찾기

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

4.3.1 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>
```

```
});  
</script>
```

4.3.2 특정 자식 노드 찾기 – children()

```
$대상노드.children("선택자")
```

선택자를 이용하여 특정 자식 노드만을 찾고 싶을 때 사용

- id가 test인 노드의 자식노드 중 className이 sel 인 노드들 border 설정

```
<script>  
$(document).ready(function() {  
    $("#test").children(".sel").css("border", "3px solid #00f");  
});  
</script>
```

4.4 부모 노드 찾기

특정 노드를 감싸고 있는 바로 위의 상위 노드를 부모 노드라고 한다

조상 노드는 특정 노드의 모든 상위 노드를 말한다

4.4.1 부모 노드 찾기 – parent()

```
$대상노드.parent()
```

특정 노드의 바로 상위에 존재하는 부모 노드를 찾을 때 사용

- id가 test인 노드의 부모 노드의 border 설정

```
<script>  
$(document).ready(function() {  
    $("#test").parent().css("border", "3px solid #f00");  
});  
</script>
```

4.4.2 조상 노드 – parents()

```
$대상노드.parents()
```

특정 노드의 모든 상위 노드, 즉 조상 노드들을 찾을 때 사용

- id가 test인 노드의 조상 노드들 border 설정

```
<script>
$(document).ready(function() {
    $("#test").parents().css("border", "3px solid #00f");
});
</script>
```

4.5 형제 노드 찾기

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

4.5.1 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 test인 노드의 이전 형제 노드 border 설정

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

4.5.2 모든 이전 형제 노드 찾기 - prevAll()

```
$대상노드.prevAll()
```

특정 노드의 모든 이전 형제 노드를 찾을 때 사용

- id가 test인 노드의 모든 이전 형제 노드들 border 설정

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

4.5.3 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() {
    $("#test").next().css("border", "2px solid #f00");
});
</script>
```

4.5.4 모든 다음 형제 노드 찾기 – nextAll()

`$대상노드.nextAll()`

특정 노드의 모든 다음 형제 노드를 찾을 때 사용

- id가 test인 노드의 모든 다음 형제 노드들 border 설정

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

5 노드 생성/추가/삭제/이동

5.1 노드 생성/추가

5.1.1 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>new div</div>");
});
</script>
```

- div 노드 생성을 자바스크립트 DOM으로 구현

```
<script>
var newDiv = document.createElement("div");
var divText = document.createTextNode("new div");
newDiv.appendChild(divText);

console.log(newDiv); // 생성된 노드 확인
</script>
```

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.

5.1.2 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

5.1.3 생성한 노드를 마지막 자식 노드로 추가

```
$부모노드.append($추가노드)
```

```
$추가노드.appendTo($부모노드)
```

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

5.1.4 생성한 노드를 특정 노드의 이전 위치에 노드 추가

```
$추가노드.insertBefore($기준노드)
```

```
$기준노드.before($추가노드)
```

신규 노드를 특정 노드의 이전 형제 노드로 추가할 경우 사용

5.1.5 생성한 노드를 특정 노드의 다음 위치에 노드 추가

```
$추가노드.insertAfter($기준노드)
```

```
$기준노드.after($추가노드)
```

신규 노드를 특정 노드의 다음 형제 노드로 추가할 경우 사용

5.2 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

5.2.1 노드를 첫 번째 자식 노드로 이동

```
$부모노드.prepend($이동노드)
```

```
$이동노드.prependTo($부모노드)
```

5.2.2 노드를 마지막 자식 노드로 이동

```
$부모노드.append($이동노드)
```

```
$이동노드.appendTo($부모노드)
```

5.2.3 생성한 노드를 특정 노드의 이전 위치로 이동

```
$이동노드.insertBefore($기준노드)
```

```
$기준노드.before($이동노드)
```

특정 노드의 이전 형제 노드로 이동시킬 경우 사용

5.2.4 생성한 노드를 특정 노드의 다음 위치로 이동

```
$이동노드.insertAfter($기준노드)
```

```
$기준노드.after($이동노드)
```

신규 노드를 특정 노드의 다음 형제 노드로 이동시킬 경우 사용

5.3 노드 삭제

5.3.1 특정 노드 삭제

```
$대상.remove()
```

찾은 노드를 삭제한다

5.3.2 모든 자식 노드 삭제

```
$대상.children().remove()
```

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

5.4 노드 내용 읽기/변경

5.4.1 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.

5.4.2 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

6 스타일 다루기

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

6.1 스타일 값 구하기

`$대상.css("스타일 속성이름")`

스타일 속성이름을 명시해 해당 스타일 속성값을 알아낸다

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

- div 노드의 border 스타일 알아내기

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

- div 노드의 width, height 스타일 알아내기

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

```

    alert("width : " + divInfo.width + ", height : " + divInfo.height);
});
</script>

```

6.2 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", "1px solid #f00");
});
</script>

```

- `#test` 노드의 `height`와 `width` 스타일 한번에 지정

```

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

```

6.3 클래스 추가

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

addClass() 메소드를 이용하여 하나 이상의 클래스를 추가할 수 있다

- #test 노드에 border 클래스 추가

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

6.4 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

7 속성

태그의 속성에는 일반 속성과 사용자 정의 속성이 있다. id, class, <a>태그의 링크 정보를 담은 href, 태그의 이미지 정보를 담은 src 등이 일반 속성이다. 그 외에 data-value와 같이 사용자가 필요에 의해 만들어서 사용하는 속성을 사용자 정의 속성이라고 부른다.

7.1 속성값 구하기

```
$대상.attr("속성이름")
```

```
$대상.data("data-속성이름")
```

일반 속성값을 구할 때에는 attr() 메소드를 사용한다

"data-"로 시작하는 속성인 사용자 정의 속성을 구할 때에는 data() 메소드를 이용하여 속성값을 구할 수 있다. 일반 속성값을 구할 때 data() 메소드를 사용하면 안 되며 "data-"로 시작하는 속성은

attr() 메소드를 속성값을 구할 수 있다.

- 속성 알아내기

```
<script>
$(document).ready(function() {
    console.log("src = " + $("img").attr("src"));
    console.log("href = " + $("a").attr("href"));
});
</script>
```

7.2 속성값 설정하기

```
$대상.attr("속성이름", "값")
$대상.data("data-속성이름", "값")
```

속성값을 알아낼 때와 마찬가지로 attr()메소드와 data() 메소드를 이용해 각각 일반 속성과 사용자 정의 속성의 속성값을 설정할 수 있다

- 속성 알아내기

```
<script>
$(document).ready(function() {
    console.log("src = " + $("img").attr("src"));
    console.log("href = " + $("a").attr("href"));
});
</script>
```