MINDs Lab

MINDs Lab Coding Convaention

for Markup Languages (HTML/CSS/Javascript)

Ver 1.1

머리글

문서 정보

문서 개요

이 문서는 마크업 개발자가 소스 코드 작성 시에 따라야 할 규칙을 기술한다.

1장. 개요

HTML, CSS, Javascript Style Guide의 필요성과 HTML, CSS, Javascript Style Guide의 요소, 용어를 소개한다.

2장. 네이밍 규칙

id/class, 이미지, 파일, 폴더의 네이밍 규칙을 설명한다.

3장. HTML 코드 작성 규칙

HTML 코드의 기본 작성 규칙과 들여쓰기, 빈 줄 사용, DTD 및 인코딩 선언, 주석 표기 규칙을 설명한다.

4장. HTML 엘리먼트 작성 규칙

HTML 엘리먼트 종류별 작성 규칙을 설명한다.

5장 CSS 코드 작성 규칙

CSS 코드의 기본 작성 규칙과 인코딩, 선택자, 속성 등의 작성 규칙을 설명한다.

6장 Javascript 코드 작성 규칙

Javascript 코드의 기본 작성 규칙 및 금지항목 및 주의사항을 설명한다.

부록 A. 코딩 시 참고 사항

웹표준 기반의 마크업, 크로스 브라우징 범위를 설명한다.

부록 B. 예약어 목록

네이밍 예약어와 대체 텍스트 예약어 목록을 설명한다.

독자

이 문서는 마크업 언어를 다루는 개발자를 대상으로 한다.

표기규칙

참고 표기

참고

독자가 참고해야 할 내용을 기술한다.

주의 표기

주의

독자가 반드시 알아야 할 사항, 시스템 에러를 유발할 수 있는 사항, 수행하지 않았을 때 재산상의 피해를 줄 수 있는 사항을 기술한다.

소스 코드 표기

이 문서에서 소스 코드는 회색 바탕에 검정색 글씨로 표기한다.

```
<div class="gnb only_pc">
    <span class="user_name"><em>홍길동</em>님</span>
    <ul class="gnb_util">
        <li><a href="my/my_use_history.html">나의 이용내역</a></li>
        <li><a href="notice/notice_list.html">공지사항</a></li>
        <li><a href="">로그아웃</a></li>
    </ul>
</div>
```

CONTENTS

CONTENTS

1. 개요

이 장에서는 Style Guide의 필요성, 요소 및 용어를 소개한다.

1.1 Style Guide 필요성

마크업 개발은 프런트-엔드 페이지의 기본 골격을 형성하기 때문에 디자인, 브라우저, 디스플레이, 스크립트, 성능, 접근성 등과 긴밀한 관계가 있다. 즉, 마크업 개발을 잘 해야 모든 브라우저 및 디스플레이에서 콘텐츠를 손실 없이, 빠르고 쉽게 사용자에게 전달할 수 있다.

Style Guide은 이러한 조건을 만족시키기 위해 마크업 개발자가 지켜야 할 표준을 제시한다.
또한, 유지보수에 투자되는 비용을 최소화하기 위해 통일된 코드 작성법을 제시한다. 코드를 최초로 작성한 사람이 끝까지 유지보수할 확률은 매우 낮다. 따라서, 최초 개발자가 아닌 사람도 코드를 빠르고 정확하게 이해할 수 있도록 작성하는 것은 코드의 유지보수 비용을 절감하고 업무 효율을 높이는 데 결정적인 역할을 한다.
적어도 한 프로젝트의 마크업 코드는 같은 Style Guide에 따라 작성해야 한다. Style Guide을 준수하면 프로젝트 멤버 간 코드 공유도 쉬워지고, 일관성 있게 코드를 작성할 수 있다. 어떤 Style Guide 을 선택하느냐가 중요한 것이 아니라, 통일된 기준으로 소스 코드를 작성하는 것이 중요하다.

1.2 Style Guide 요소

1. 네이밍 규칙

네이밍 규칙은 레이아웃, 객체, 이미지, 폴더, 파일의 이름을 작성하는 규칙이다. 이해하기 쉬운 이름으로 작성해야 코드를 쉽게 파악할 수 있다. 단, 모바일과 같이 성능, 네트워크 속도를 고려해야 하는 경우 축약된 형태로 작성할 수 있다.

2. HTML 코드 작성 규칙

A. 들여쓰기

HTML 코드를 작성할 때 코드의 가독성을 높이기 위하여 왼쪽 첫 번째 열부터 오른쪽으로 일정한 간격만큼 띄어 쓴다. 들여쓰기를 하면 전체 HTML 구조를 쉽게 파악할 수 있다.

B. 빈 줄

HTML 코드의 빈 줄은 코드 그룹의 영역을 표시하기 위하여 사용한다.

C. DTD 및 인코딩

DTD(Document Type Definition)는 SGML(Standard Generalized Markup Language) 계열 마크업 언어의 문서 타입을 정의하는 것으로서, 해당 HTML 문서가 어떤 버전의 HTML로 작성되었는지, 어떤 규칙으로 내용을 기술하고 어떤 엘리먼트와 애트리뷰트, 애트리뷰트값을 지정할 수 있는지 정의한다. 또한 인코딩을 선언하여 문서에서 사용되는 문자 코드 세트를 지정한다. DTD와 인코딩 선언은 HTML 문서가 브라우저에서 바르게 해석될 수 있도록 한다.

D. 주석

HTML 코드의 주석은 코드 그룹을 구분하거나, 코드의 수정과 삭제를 표시하거나, 개발자가 참고해야 하는 사항을 기술한다.

3. HTML 엘리먼트 작성 규칙

HTML 엘리먼트 작성 규칙은 반드시 선언해야 하는 애트리뷰트와 선택적 사용이 가능한 애트리뷰트에 대한 내용을 기술하고 선언 순서를 제시하여 코드 품질을 유지한다.

1.2 Style Guide 요소

4. CSS 코드 작성 규칙(1/2)

A. 들여쓰기

CSS 코드는 들여쓰기를 하지 않는다.

B. 공백

CSS 코드는 공백을 최소화한다.

C. 빈 줄

CSS 코드의 빈 줄은 코드 그룹의 영역을 표시하기 위하여 사용한다.

D. 줄 바꿈

CSS 코드의 가독성을 위한 줄 바꿈은 하지 않는다..

E. 인코딩

CSS의 인코딩은 HTML의 인코딩과 동일하게 선언하여 HTML 문서가 브라우저에서 바르게 해석될 수 있도록 한다.

F. 선택자

선택자 버그가 발생하지 않도록 사용 규칙을 준수한다.

G. 속성

속성 선언 순서를 준수하여 개발자가 코드를 쉽게 파악할 수 있도록 하며, CSS 코드 최적화를 위해 속성 값을 축약하여 사용하고 약식 속성을 허용 범위에 맞게 사용한다.

H. z-index

z-index 속성 값을 범위에 맞게 사용하여 객체가 브라우저에서 바르게 표현되도록 한다.

1.2 Style Guide 요소

4. CSS 코드 작성 규칙(2/2)

I. 핵

크로스 브라우징을 위해 제시된 핵(hack)에 한해 최소한의 사용을 허용한다.

I. CSS 선언 타입

상황에 알맞은 CSS 선언 타입을 선택한다. .

J. 주석

CSS 코드의 주석은 코드 그룹을 구분하거나, 코드의 수정과 삭제를 표시하거나, 개발자가 참고해야 하는 사항을 기술한다.

6. 웹 접근성 보장 방법

모든 사람이 환경의 제약 없이 웹 콘텐츠에 접근할 수 있도록 보장하는 마크업 방법을 기술한다.

1.3 Style Guide 용어

A. 엘리먼트(Element)

HTML 문서를 구성하는 요소를 의미한다. 일반적으로 시작 태그, 내용, 종료 태그로 구성된다.

B. 애트리뷰트(attribute)

엘리먼트에 부여할 수 있는 특성을 의미한다. 기본값이 설정되어 있으나 애트리뷰트를 선언하여 다른 값으로 설정할 수 있다.

C. 선택자(selector)

엘리먼트에 CSS 스타일을 적용하기 위한 패턴이다. 이 문서에서 언급하는 선택자의 종류는 다음 표와 표 1-1같다.

표 1-1 선택자 종류

이름	패던 예	설명
공통 선택자	*	어떤 엘리먼트와도 일치함.
타입 선택자	div	해당 엘리먼트와 일치함.
하위 선택자	div p	해당 엘리먼트의 하위의 모든 엘리먼트와 일치함.
자식 선택자	div > p	해당 엘리먼트의 자식 엘리먼트와 일치함.
Class 선택자	div.class	해당 엘리먼트의 class 애트리뷰트값과 일치함.
Id 선택자	div#id	해당 엘리먼트의 id 애트리뷰트값과 일치함.

D. 속성(property)

엘리먼트에 부여할 CSS 스타일 특성을 의미한다. 기본값이 설정되어 있으나 속성을 선언하여 다른 값으로 설정할 수 있다. 각 속성 단위는 세미콜론(;)으로 구분지어 선언한다.

1.3 Style Guide 용어

E. 속성 값(value)

속성에 부여하는 값으로 콜론(:)으로 구분지어 선언한다

F. 예약어

이 문서에서 사용하는 예약어는 일반적인 의미와 다르게 사용된다. 이 문서에서 예약어는 객체, 이미지, 파일 및 폴더의 네이밍 시 의미를 일관되게 표현하기 위해 미리 지정해 놓은 일종의 언어 규칙을 의미한다. 예를 들어, '검색'을 표현하는 예약어가 'srch'일 경우, 검색 영역을 위한 객체의 class 이름은 'srch'라는 예약어를 반드시 포함해야 한다.

2. 네이밍 규칙

이 장에서는 id/class, 이미지, 파일, 폴더의 네이밍 규칙을 설명한다.

2.1 기본 규칙

다음과 같은 기본 네이밍 규칙을 준수한다.

A. 일반 규칙

이름은 영문 소문자, 숫자, 언더스코어(_)로 작성한다.

B. 시작 이름

이름은 영문 대문자, 숫자로 시작할 수 없다.

표 2-1 공통 네이밍 규칙 예

잘못된 예	올바른 예
Btn	btn
2btn	btn02

C. 예약어

- 예약어가 있는 경우 예약어를 사용한다.
- 예약어는 표 B-1 공통 네이밍 예약어에 근거하여 작성한다.
- 예약어가 없으면, 종류와 특성을 나타내도록 네이밍한다.

D. 영문 소문자, 숫자, 언더스코어 조합

- 영문 소문자, 숫자, 언더스코어(_)를 사용할 수 있다.
- 언더스코어(_)는 2개 이상의 단어를 조합할 때 사용한다.
- 단어와 숫자를 조합하는 경우 언더스코어(_)를 생략한다.
- 언더스코어(_)가 포함된 예약어는 숫자, 영문 소문자와 조합하여 사용할 수 있다.
- 언더스코어(_)를 이용하여 3단계를 초과하여 조합할 수 없다.
- 3단계를 초과 할 경우 조합어를 사용하고, 조합어의 첫 문자는 대문자로 표기한다.

표 2-2 레이아웃 예약어 범위

기본형	잘못된 예	올바른 예
section	sectionlist	stnLst, stn_lst
	section_list_style_type	stn_lst_type
		stn, section1, section2
no		no1, n02

2.2 id 및 class 네이밍 규칙

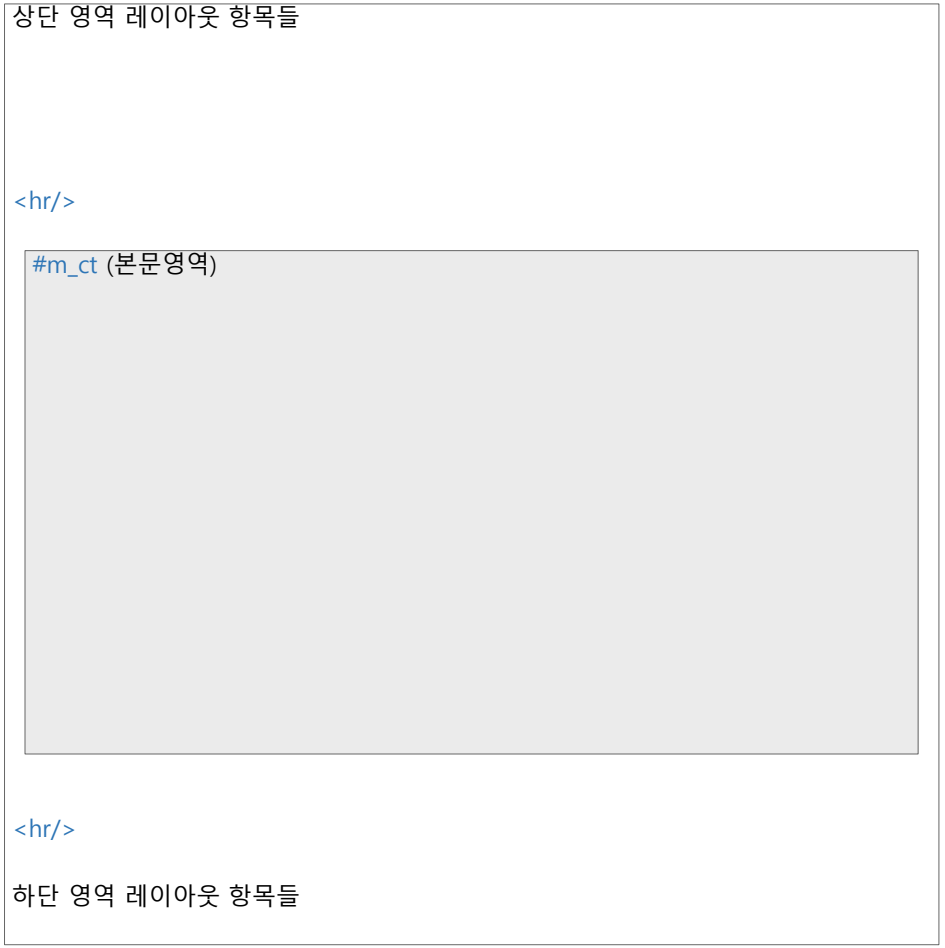
다음 그림은 HTML의 기본 템플릿 구조를 나타낸 것이다.

Pc (main)

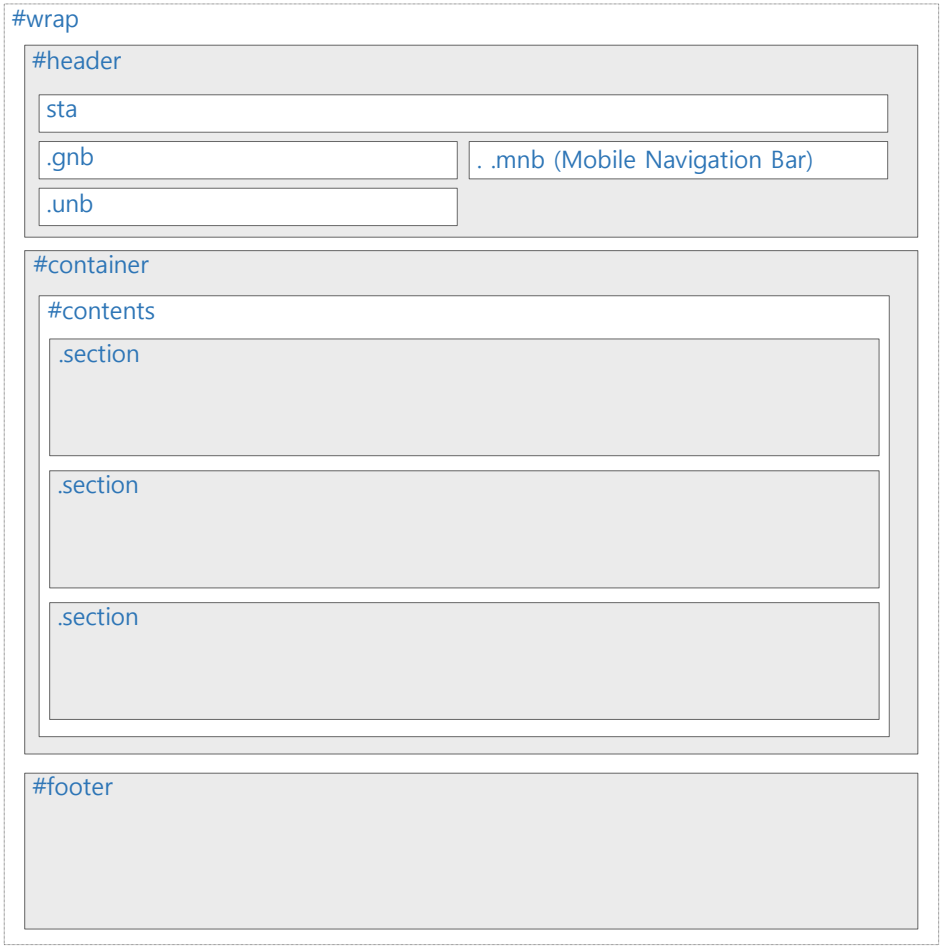
Pc (sub)

2.2 id 및 class 네이밍 규칙

다음 그림은 HTML의 기본 템플릿 구조를 나타낸 것이다.



Mobile



Responsive Web

2.2 id 및 class 네이밍 규칙

A. id, class

- id는 문서 전체의 고유 식별자 이므로 한 문서에서 동일한 id를 여러 번 사용하지 않는다.
- 레이아웃을 제외한 id는 스타일을 지정하지 않는다.
- class는 문서에서 여러 번 사용할 수 있다.

B. 레이아웃 예약어

레이아웃에는 다음 표에 예약된 id를 사용한다.

표 2-3 레이아웃 예약어 범위

예약어	범위
#wrap	페이지 전체 영역
#header	머리글 영역
#container	본문영역
#contents	주요 콘텐츠 영역
#footer	바닥글 영역

2.2 id 및 class 네이밍 규칙

C. 팝업 레이아웃 예약어

- 팝업 문서의 레이아웃 지정 범위는 동일하다.
- 레이아웃 예약어 앞에 'pop_'를 조합하여 사용한다.

표 2-4 팝업 레이아웃 예약어 범위

예약어	범위
#pop_wrap	페이지 전체 영역
#pop_top	머리글 영역
#pop_mid	본문영역
#pop_ct	주요 콘텐츠 영역
#pop_btm	바닥글 영역

D. 레이아웃 네이밍 조합

- 레이아웃 id의 네이밍은 조합하여 사용할 수 없다.
- 레이아웃에 디자인 속성을 추가/변경하려면 class를 사용한다.

표 2-5 레이아웃 네이밍 조합 예

잘못된 예	올바른 예
#wrap2,	#wrap

```
<div id="wrap">
    <div id="header"></div>
    <div id="container" class="container_type">
        <div id="content"></div>
    </div>
```

2.2 id 및 class 네이밍 규칙

E. 객체 네이밍

- 객체 예약어는 표 B-2 객체 예약어에 근거하여 작성한다.
- 객체는 class만 사용할 수 있으며, 전사 공통 마크업 템플릿의 class와 중복되지 않아야 한다.
- 개발과 연동되는 동적 객체도 class만 사용한다.
- 팝업, iframe에도 동일한 규칙을 적용한다.

F. class 네이밍 확장

- 종속 확장 class: 기본형 class에 종속되어 여백, 색깔, 행간 등의 몇 가지 속성을 부여하고자 할 때 사용하는 class.
- 독립 확장 class: 기본형 class의 변형이 타입으로 분류할 만큼 클 경우 사용하는 class.

표 2-6 class 네이밍 확장 예

기본형	확장형	설명
guide_lst	guide_lst01, guide_lst02	독립 확장 class
	guide_lst type01, guide_lst type02	종속 확장 class

2.3 이미지 네이밍 규칙

다음과 같은 기본 네이밍 규칙을 준수한다.

A. 이미지 네이밍

- 이미지 예약어는 표 B-3 이미지 예약어에 근거하여 작성한다.
- 같은 분류의 이미지가 두 개 이상이면 파일 이름 마지막에 숫자를 추가하여 구분한다. 단, 파일 이름마지막에 숫자 정보가 포함되어 있을 경우 _v숫자를 추하여 구분한다. 이 경우에만 3 단계 초과 조합을 허용한다.
- 임시 이미지에 한해 특수문자를 사용할 수 있다.
- 이미지 네이밍은 이미지 확장자와 관계 없이 순차적으로 적용한다. 예) bu_dot1.gif, bu_dot2.jpg, bu_dot3.png
- 이미지 네이밍을 확장해야 할 때는 B. 이미지 네이밍 조합 규칙을 준수한다.
- 임시 이미지에 한해 @네이밍을 허용한다. 예) @tmp_

B. 이미지 네이밍 조합

이미지 이름은 '형태_의미_상태' 순서로 조합한다

표 2-7 이미지 네이밍 조합 예

잘못된 예	올바른 예	설명
on_recommend_tab1	tab1_recomm_on	'형태_의미_상태' 순서로 조합한다.
bnm.gif	btn_popup_mail.gif	임의로 축약하지 않는다.
btn_search_popup_mail.gif	btn_srch_mail.gif	3단계 이하로 조합한다.
Ico_num_black1_1	ic_num_black1_v1.gif	숫자 정보 포함 시 3 단계 초과 조합을 허용한다.
btn_Search.gif	btn_srch.gif	영문 소문자를 사용한다.
1btn_search.gif	btn_srch.gif	숫자로 시작하지 않는다

2.4 파일 및 폴더 네이밍 규칙

다음과 같은 파일 및 폴더 네이밍 규칙을 준수한다.

A. 파일 및 폴더 네이밍

- 산출되는 모든 HTML 파일과 폴더를 분류해야 할 경우, 페이지 마스터 파일에서 정의된 페이지 코드와 폴더 코드를 그대로 사용한다. 단, '코드_이름' 규칙 중 마크업 산출물의 경우만 코드만으로 파일 명을 정의할 수 있으며 뒤의 '이름(영문만 허용) '은 선택에 따라 정의할 수 있다.
- 정의된 코드가 없는 프로젝트의 경우, 표 2-8 파일 및 폴더 네이밍 규칙을 따른다.
- 모바일의 경우 서비스 및 디바이스 환경에 따라 변경될 수 있으므로 표 2-9 를 참조한다.

표 2-8 파일 및 폴더 네이밍 규칙 예

분류	예제	설명
HTML	1.1.1_home.html	'페이지코드.html'로 사용하거나, 선택에 따라 '페이지 코드_이름(영문만 허용).html'로 사용한다.
HTML (페이지 코드 정의서가 없는 경우)	news.html	'서비스영문이름.html' 로 사용
	pop_.html	팝업 파일을 사용
	ifr_.html	iframe 파일을 사용
CSS	news_.css	'서비스영문이름.css'로 사용한다.
folder	p_yymmdd_프로젝트이름	신규 프로젝트 작업 시 사용
	yymmdd_서스테이닝이름	서스테이닝 작업 시 사용
	img, css, javascript	images, css, js 폴더 사용
	p_yymmdd_프로젝트명₩00_메뉴명	HTML 파일의 폴더 분류가 필요한 경우 사용

2.4 파일 및 폴더 네이밍 규칙

B. **파일 및 폴더 네이밍 조합**

- HTML 파일은 페이지 마스터 파일에서 정의한 페이지 코드를 그대로 사용하거나, '코드_이름(영문만 허용)'으로 조합한다.
- 페이지 마스터 파일이 없는 경우 '메뉴이름_의미_상태' 순서로 조합한다.
- CSS 파일은 '서비스이름'을 맨 앞으로 하여 조합한다.

표 2-10 HTML/CSS 조합 예

분류	조합 예	설명
HTML	cafe.html	' 페이지 마스터 파일에서 정의할 페이지 코드를 그대로 사용하거나, '코드_네임(영문만 허용)'으로 조합한다
	cafe_menu_detail.html	
	news_nboard_view.html	정의된 코드가 없는 경우 '메뉴이름_의미_상태' 순서로 조합한다.
CSS	news_home.css	'서비스이름'을 맨 앞으로 하여 조합한다.
	news_admin.css	
	news_pop.css, news_nanum.css	

3. HTML 코드 작성 규칙

이 장에서는 HTML 코드의 기본 작성 규칙과 들여쓰기, 빈 줄 사용, DTD 및 인코딩 선언, 주석 표기 규칙을 설명한다.

3.1 기본 규칙

HTML 코드를 작성할 때 다음과 같은 기본 규칙을 준수한다.

A. W3C Validation

HTML은 해당 DTD의 명세에 맞게 작성하며, W3C validation을 통과해야 한다. 단, HTML5 DTD 선언 시 다음 오류 내용은 허용한다.

- <iframe>의 frameborder 애트리뷰트

B. 영문 소문자 사용

DTD를 제외한 모든 엘리먼트와 애트리뷰트는 소문자로 작성한다.

표 3-1 소문자 작성 예

잘못된 예	올바른 예
<SPAN Class="desc">간단한 설명</SPAN>	간단한 설명

C. 애트리뷰트값 표기

애트리뷰트값은 큰따옴표(" ")로 묶는다.

표 3-2 애트리뷰트값 표기 예

잘못된 예	올바른 예
<img src='test.gif' width='100' height='100' alt='테스트'>	<img src="test.gif" width="100" height="100" alt="테스트">

D. Character entity references 사용

특수 기호는 문자 엔티티 참조를 사용하여 코드로 변환한다. 자세한 사항은 아래 경로에서 확인할 수 있다. 단, 아포스트로피(')는 '로 선언한다.

HTML 4.01의 Character entity references: http://www.w3.org/TR/html4/sgml/entities.html

HTML 5의 Character references: http://www.w3.org/TR/html5/syntax.html#character-references

표 3-3 Character entity references 사용 예

잘못된 예	올바른 예
<a href="...&nid=2">	<a href="... &nid=2">

참고

<, >, ", ', & 등의 특수기호를 Character entity references로 변환하지 않으면, 브라우저가 이를 시작/종료 엘리먼트나 애트리뷰트로 잘못 해석할 수 있다. 자동으로 생성되는 링크의 경로나 이미지의 alt값에도 Character entity references가 바르게 적용되도록 한다.

E. 스크립트 선언 지양

Jindo Component를 제외한 이벤트 핸들러 및 스크립트는 HTML 산출물에 선언하지 않는다.

표 3-4 스크립트 선언 예

잘못된 예	올바른 예
<input type="text" id="user_id" name="user_id" onfocus="this.className='focus'">	<!-- [D] 입력 박스에 포커싱되었을 때, class="focus" 추가 --> <input type="text" name="user_id" id="user_id" class="focus">

3.2 들여쓰기 규칙

들여쓰기를 하면 코드의 가독성이 높아지고 전체 HTML 구조를 쉽게 파악할 수 있다. 다음과 같은 들여쓰기 규칙을 준수한다.
마크업의 중첩이 깊어질 때마다 자식 엘리먼트는 1탭 들여 쓰고, 1탭의 크기는 공백 4칸으로 설정한다. 문서 내에서 반드시 탭을 이용하여 들여쓰기를 하며, 탭을 대쉬하여 공백으로 띄어 들여쓰지 않는다.

표 3-5 들여쓰기 사용 예

잘못된 예	올바른 예
<table> <caption>...</caption> <colgroup> <col>...</col> </colgroup> <thead> <tr> <th>...</th> <th>...</th> </tr> </thead> <tbody> <tr> <td>...</td> <td>...</td> </tr> </tbody> </table>	<table> <caption>...</caption> <colgroup> <col>...</col> </colgroup> <thead> <tr> <th>...</th> <th>...</th> </tr> </thead> <tbody> <tr> <td>...</td> <td>...</td> </tr> </tbody> </table>
   	 ... ...

3.2 들여쓰기 규칙

잘못된 예	올바른 예
<dl> <dt>...</dt> <dd>...</dd> </dl>	<dl> <dt>...</dt> <dd>...</dd> </dl>

참고

Dreamweaver와 Editplus에서 Tab Size 설정하는 법

- Dreamweaver: [Preferences] > [Category] > [Code Format] > [Indent & Tab Size] > Indent with 1 Tabs, Tab size 4
- Editplus: [Tools] > [Preferences] > [Categories] > [Files] > [Settings and syntax] > [Tab/Indent]

3.3 빈 줄

빈 줄을 사용하려면 다음과 같은 사용 규칙을 준수한다. 의미 있는 객체를 구분하기 위하여 코드 그룹 간 1줄씩 빈 줄을 만드는 것은 허용한다. 빈 줄의 간격은 1줄을 초과하지 않는다.

빈 줄을 사용하는 것은 선택 사항이다.

표 3-6 빈줄 사용 예

잘못된 예	올바른 예
<head> 내용... </head> 빈 줄 빈 줄 <body> 내용... </body>	<head> 내용... </head> 빈 줄 <body> 내용... </body>

3.4 DTD 및 인코딩

1. DTD 선언(1/2)

HTML 문서는 반드시 DTD를 선언한다.

A. 기본 DTD

신규 HTML 문서를 작성할 때 'HTML5'를 사용한다. 작성 예는 다음과 같다. 모바일 동일.

```
<!DOCTYPE html>
```

참고

HTML5는 SGML에 기본으로 하지 않기 때문에, DTD의 참조를 요하지 않는다.
<!DOCTYPE>은 웹 브라우저가 HTML이 무슨 버전인지 알수 있게 해준다.

B. 기타 DTD

사용 중인 서비스를 부분 개편하거나 완전히 개편하더라도 기존의 HTML/CSS 데이터 의존도가 높다면, 기준과 동일한 DTD를 사용할 수 있다. 작성 예는 다음과 같다.

HTML 4.01 Transitional

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

XHTML 1.0 Transitional

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

Quirks Mode

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
```

3.4 DTD 및 인코딩

3.4.1 DTD 선언(2/2)

주의

아래와 같은 경우 DTD 설정별 표준 문법으로 마크업 하더라도 브라우저에서 Quirks Mode로 인식하여 바르게 해석되지 않으므로 주의한다.

DTD가 선언되지 않은 경우(html 태그로 문서 시작)

```
<html>
```

선언한 DTD 앞에 다른 문자가 오는 경우

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

3.4 DTD 및 인코딩

2. 인코딩 선언

HTML 문서는 반드시 인코딩 정보를 선언한다. 인코딩 설정은 DB의 인코딩 방식과도 관련이 있으므로 반드시 담당 개발자와 협의하여 정해야 한다.

A. 기본 인코딩

신규 HTML 문서를 작성할 때 기본 인코딩은 utf-8을 원칙으로 한다. 다음은 인코딩 방식으로 utf-8을 선언한 예이다.

```
<meta http-equiv="Content-Type" content="text/html;charset=utf-8">
```

HTML5 DTD 사용 시에는 아래와 같이 선언한다.

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

utf-8은 다국어 지원이 가능하며, euc-kr보다 표현 가능한 한글(고어, 음절 등)이 더 많다.

참고

한글은 utf-8에서 3바이트, euc-kr에서 2바이트를 차지하기 때문에, utf-8이 euc-kr에 비하여 DB 저장 용량이나 트래픽이 많을 수 있다.

Internet Explorer 7 이상의 버전에서 아래와 같이 링크의 밑줄이 글자와 붙어 보이는 현상이 있다.

무궁화꽃이피었습니다.

B. 기타 인코딩

utf-8 인코딩을 사용할 수 없으면 euc-kr을 사용한다. 다음은 인코딩 방식으로 euc-kr을 선언한 예이다.

```
<meta http-equiv="Content-Type" content="text/html;charset=euc-kr">
```

HTML, CSS 파일을 저장할 때 반드시 설정한 인코딩을 선택하여 저장한다.

3.4 DTD 및 인코딩

3.4.3 Viewport

모바일 브라우저에 대응하는 HTML 문서의 <head> 앞에 반드시 뷰포트를 설정한다.
뷰포트는 브라우저와 해상도에 따라 다르게 설정한다. 아래는 브라우저에 따른 뷰포트 설정 예이다.

표 3-7 브라우저에 따른 뷰포트 설정

브라우저	뷰포트
사파리	<meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0, user-scalable=no" />
안드로이드 돌핀	meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0, user-scalable=no, target-densitydpi=medium-dpi" />
오페라 모바일 9.5	<meta name="viewport" content="initial-scale=0.75, maximum-scale=0.75, minimum-scale=0.75, user-scalable=no" />
기타(default) 폴라리스, LGT 웹뷰어	<meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0, user-scalable=no, target-densitydpi=medium-dpi" />

참고 Viewport란?

웹 페이지 전체 중 브라우저 창에 보이는 부분을 말하며, 창 크기를 조절하여 뷰포트를 크게 또는 작게 만들 수 있다.
그러나 대부분의 모바일 브라우저는 창 크기 조절 기능이 없고, 기기마다 해상도도 다르기 때문에 뷰포트 속성을 사용하여 웹 페이지 전체 크기와 창에 보이는 부분을 설정할 수 있다. 폴라리스6, 오페라 모바일 8.65 등 뷰포트 속성을 지원하지 않는 브라우저도 있다.

3.5 주석

A. 기본 형식

HTML 주석의 시작과 종료는 아래와 같이 표기하며, 기본 형식에 맞게 작성한다.

```
시작 주석      <!-- #주석내용  -->
종료 주석      <!-- //#주석내용  -->
```

- 주석 기호와 주석 내용 사이에는 반드시 공백 두 칸이 있어야 한다.
- 주석 기호와 주석 내용 사이의 줄 바꿈은 허용하지 않는다. 단, 주석 내용 긴 줄 바꿈은 허용한다.
- 주석 내용 앞에는 id와 class 구분을 표시한다.

표 3-8 주석 기본 형식 예

잘못된 예	올바른 예
<!--GNB--> <!--* GNB *--> <!-- GNB --> 빈줄 ... <!-- //.GNB-->	<!-- gnb--> ... <!-- //.gnb -->

주의
마크업과 개발의 편의를 위해 작성한 주석은 실제 서비스를 적용할 때 반드시 삭제한다.

3.5 주석

B. 레이아웃 및 콘텐츠 영역의 주석 표기

wrap을 제외한 레이아웃과 독립된 콘텐츠 영역의 끝에 주석을 표기하며, 레이아웃은 id 이름과 동일하게 주석을 넣는다. 독립된 콘텐츠 영역의 주석 표기는 선택 사항이다.

HTML 코드와 주석은 줄로 분리해야 한다.

```
<!-- #content -->
<div id="content">
    <!-- .namecard  -->
    <div class="namecard"> ... </div>
    <!-- //.namecard -->
</div>
<!-- //#content -->
```

B. **개발 적용과 관련된 주석 표기**

개발 적용과 관련된 주석은 해당되는 영역 위에 표기하며, 종료 주석은 표기하지 않는다. 주석 앞에는 [D]라는 말머리를 사용하여 담당 개발자가 반드시 확인할 수 있도록 한다. 주석이 두 줄 이상이 되더라도 주석 기호는 한 번만 표기한다.

잘못된 예	올바른 예
<!-- 케이스별 클래스 변화 --> <!-- 의사 : my_doctor 변호사 : my_lawyer -->	<!-- [D] 케이스별 클래스 변화 의사 : my_doctor 변호 사 : my_lawyer -->
<img src="btn.gif" width="100" height="100" alt="버튼"> <!-- 숨김처리 <img src="btn_on.gif" width="100" height="100" alt="버튼 "> //숨김처리 -->	<img src="btn.gif" width="100" height="100" alt="버튼"> <!-- [D] 활성된 버튼은 파일명에 _on추가 <img src="btn_on.gif" width="100" height="100" alt="버튼"> -->

3.6 초기파일

신규 HTML 문서를 작성할 때 아래 파일을 기본으로 한다.

```
<!DOCTYPE html>
<html lang="ko">
<head>
<meta charset="utf-8">
<title>메뉴 :: 브랜드명 서비스</title>
<link rel="stylesheet" type="text/css" href="sevice_name.css">
</head>
<body>
<!-- #wrap -->
<div id="wrap">
    <!-- #header -->
    <div id="header">
    </div>
    <!-- //#header -->
    <!-- //#container -->
    <div id="container">
        <!-- content -->
        <div id="content">
        </div>
        <!-- //#content -->
    </div>
    <!-- //#container -->

    <!-- # footer -->
    <div id="footer">
    </div>
    <!-- //# footer -->
</div>
<!-- //#wrap -->
</body>
</html>
```

3.6 초기파일

모바일 신규 HTML 문서를 작성할 때 아래 파일을 기본으로 한다.

```
<!doctype html>
<html lang="ko">
<head>
<meta charset="utf-8">
<meta name="viewport" content="..">
<title>상단: 헤더명</title>
<link rel="stylesheet" type="text/css" href="css/..">
</head>
<body class="..">
    <div class="u_skip"><a href="#ct">본문 바로가기</a></div>

     <!-- header -->
    <header>
    </header>
    <!-- //header -->

    <!-- #ct -->
    <div id="m_ct" role="main">
    </div>
    <!-- //#ct -->

     <!-- footer -->
    < footer >
    </ footer >
    <!-- //footer -->
</body>
</html>
```

4. HTML 엘리먼트 작성 규칙

이 장에서는 HTML 엘리먼트 종류별 작성 규칙을 설명한다.

4.1 기본 규칙

특정 엘리먼트에 class, style을 선언할 때는 선언 순서를 준수한다. 다음과 같이 class와 style은 제일 뒷부분에 선언한다.

```
<input type="text" id="user_id" title="사용자ID" class="input_txt" style="width:100px">
```

4.2 전역 구조화 엘리먼트

A. <html>

다음과 같이 lang 애트리뷰트를 선언한다.

class 애트리뷰트는 선언하지 않는다.

```
<html lang="ko">
```

XHTML DTD 선언 시에는 다음과 같이 lang 애트리뷰트를 선언한다.

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="ko" lang="ko">
```

lang 애트리뷰트는 User Agent가 언어를 올바로 해석할 수 있게 도와주며, 검색과 음성 장치(speech synthesizers)에 활용된다. 언어 코드는 모든 엘리먼트에 사용할 수 있지만 HTML 엘리먼트에 해당 문서의 주 언어 코드만 선언한다.

HTML 4.01 Specification, XHTML 1.0 Specification에서 자연어는 RFC 1766에 명시된 언어 코드를 사용한다. RFC1766은 각각 ISO639, ISO3166을 참조하며, ISO Specification은 유료이므로 아래 웹 페이지를 참조한다.

• MSDN Language Codes: http://msdn.microsoft.com/en-au/library/ms533052(VS.85).aspx

자주 사용하는 국가코드는 en(영어), ja(일본어), zh-cn(중국어)이다.

B. <head>

meta, title, link, script, style 순서로 엘리먼트를 선언한다.

```
<head>
<meta http-equiv="Content-Type" content="text/html;charset=utf-8">
<meta http-equiv="Content-Script-Type" content="text/javascript">
<meta http-equiv="Content-Style-Type" content="text/css">
<title>소개 :: 마음에이아이</title>
<link rel="stylesheet" type="text/css" href="css/default.css">
<script type="text/javascript" src="js/default.js"></script>
<style type="text/css">
[stuff]
</style>
</head>
```

4.2 전역 구조화 엘리먼트

C. <meta>

문서의 기본 인코딩, 뷰포트, 스크립트 형식, 스타일 형식 순서로 엘리먼트를 선언한다.

뷰포트는 모바일 브라우저에 대응하는 HTML의 경우에만 선언한다.

```
<meta http-equiv="Content-Type" content="text/html;charset=utf-8">
<meta name="viewport" content="width=device-width,initial-scale=1.0,maximum-scale=1.0,minimum-scale=1.0,user-scalable=no,target-densitydpi=medium-dpi">
<meta http-equiv="Content-Script-Type" content="text/javascript">
<meta http-equiv="Content-Style-Type" content="text/css">
```

HTML5 DTD 선언 시에는 다음과 같다.

```
<meta charset="utf-8">
<meta name="viewport" content="width=device-width,initial-scale=1.0,maximum-scale=1.0,minimum-scale=1.0,user-scalable=no,target-densitydpi=medium-dpi">
```

D. <title>

"메뉴 :: 브랜드명 서비스"의 형식으로 작성한다.

```
<title>소개 :: 마음에이아이</title>
```

E. <link>

rel, type, href 순서로 애트리뷰트를 선언한다.

```
<link rel="stylesheet" type="text/css" href="css/default.css">
```

F. <script>

type, src 순서로 애트리뷰트를 선언한다.

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

script는 <head></head> 내에 선언하는 것을 원칙으로 한다. 단, 성능상의 이슈가 있으면 개발 부서와 협의하여 선언 위치를 변경한다.

참고

language 애트리뷰트는 HTML4 이전 버전의 하위 호환성을 위해 사용하는 비표준 애트리뷰트로, 사용하지 않는다.

HTML5 문서의 경우 <script>와 <style> 선언 시 type 애트리뷰트의 생략을 허용한다.

4.2 전역 구조화 엘리먼트

G. <style>

type 애트리뷰트를 선언한다.

```
<style type="text/css">
    [stuff]
</style>
```

H. <body>

class 애트리뷰트는 웹페이지의 스킨셋을 변경해야 할 때 선택적으로 사용한다.
모바일에서는 <body>에 브라우저별로 정의된 class명을 지정하고 이를 상속하여 브라우저별 CSS 렌더링 차이를 해결한다. 변경 사항이 잦으니 아래 경로에서 확인한다.

표 4-1 <body> 태그의 클래스 추가

브라우저	클래스
사파리	s, s2(iOS 4.2 미만)
안드로이드	a
돌핀	d
오페라 모바일 9.5	o

```
<body class="s">
```

4.3 폼 엘리먼트

폰 컨트롤 엘리먼트를 마크업할 때 <form>, <fieldset>, <legend> 엘리먼트를 다음과 같이 선언한다. 단, 필요에 따라 개별적으로 사용할 수 있다.

```
<form>
<fieldset>
<legend>개인정보</legend>
    [form control element here]
</fieldset>
</form>
```

A. <fieldset>

<form> 엘리먼트의 자식 노드로 선언하여 폰 컨트롤 엘리먼트들을 그루핑하기 위해 선언한다.

```
<form>
<fieldset>
<legend>개인정보</legend>
    [stuff]
</fieldset>
</form>
```

B. <legend>

폰 컨트롤 그룹인 <fieldset>의 자식 엘리먼트로서 폰 컨트롤 엘리먼트들의 그룹 이름을 표현하기 위해 선언한다.

```
<form>
<fieldset>
<legend>개인정보</legend>
    [stuff]
</fieldset>
</form>
```

4.3 폼 엘리먼트

C. <input>

<label> 엘리먼트, title 애트리뷰트, alt 애트리뷰트를 통해 스크린 리더 사용자는 주변 맥락에 대한 이해 없이 각 엘리먼트에 독립적으로 접근해도 폰을 이해할 수 있다. 웹 접근성 항목은 6. 웹 접근성 보장 방법을 참조한다.

```
<label for="user_id">아이디</label> <input type="text" id="user_id" name="user_id">
<input type="image" src="btn_confirm.gif" alt="확인">
<label for="num1">유선전화</label>
<input type="text" id="num1" name="num1" title="지역번호">
```

다음과 같이 type값에 따라 애트리뷰트를 선언한다.

- type이 text인 경우: type, id, title, value, accesskey 순서로 애트리뷰트를 선언한다.

```
<input type="text" id="user_id" title="사용자ID" value="아이디를 입력하세요" accesskey="I">
```

  - 동일한 스타일의 텍스트필드이나 너비값이 다를 경우 style 애트리뷰트를 이용하여 width값을 제어한다.
  - 텍스트필드에 가이드를 위한 내용이 들어갈 경우 value 애트리뷰트를 선택적으로 사용할 수 있다.

- 1type이 radio, checkbox일 경우: type, name, id, checked 순서로 애트리뷰트를 선언한다.

```
<input type="radio" name="vt_align" id="align_lft" checked="checked"><label for="align_lft">왼쪽정렬</label>
<input type="radio" name="vt_align" id="align_cnt"> <label for="align_cnt">가욲데정렬</label>
<input type="radio" name="vt_align" id="align_rgt"> <label for="align_rgt">오른쪽정렬</label>

<input type="checkbox" name="sports" id="soccer" checked="checked"> <label for="soccer">축구</label>
<input type="checkbox" name="sports" id="basketball"> <label for="basketball">농구</label>
<input type="checkbox" name="sports" id="tennis"> <label for="tennis">테니스</label>
```

  - 그루핑이 필요하면 선택적으로 name 애트리뷰트를 이용하여 항목들을 그루핑한다.
  - 기본 선택에 대한 표현이 필요할경우 checked="checked"라고 표기한다.

- type이 image일 경우: type, src, alt 순서로 애트리뷰트를 선언한다.
  - alt 애트리뷰트를 반드시 선언한다.

```
<input type="image" src="img/btn_confirm.gif" alt="확인">
```

4.3 폼 엘리먼트

- type 이 file일 경우: type 애트리뷰트를 선언한다.

```
<input type="file">
```

- type 이 button, reset, submit 일 경우: type 애트리뷰트를 선언한다.

```
<input type="button">
```

D. <select>

동일한 스타일의 셀렉트 박스이나 너비값이 다르면 선택적으로 style 애트리뷰트를 이용하여 width값을 제어한다.

```
<select (id=" ") (title=" ") style="width:100px">
    <option>등급</option>
</select>
```

E. <label>

<input>(text, checkbox, radio, file, password), <select>, <textarea>와 같은 폰 엘리먼트는 for 애트리뷰트를 부여하여 해당 엘리먼트의 id값과 동일한 이름으로 연결(coupling)한다. 단, 레이블 명이 위치한 공간이 없는 경우 title 애트리뷰트로 대체할 수 있다.

```
<input type="radio" name="alignment" id="align_left"> <label for="align_lft">왼쪽정렧</label>
<input type="checkbox" name="sports" id="soccer"> <label for="soccer">축구</label>
```

단, <label> 안에 <input> 앨리먼트가 있는 경우 for와 id를 이용하여 연결하지 않아도 된다.

```
<label><input type="checkbox" name="sports">축구</label>
```

F. <textarea>

cols, rows 순서로 애트리뷰트를 선언한다.

CSS를 정상적으로 불러오지 못하는 상황에서도 사용에 문제가 없도록 col, rows의 애트리뷰트값은 각각 최소 30, 5 이상이 되도록 선언한다. <label>로 커플링 할 수 없는 경우 title을 사용하여도 무방하며 id, title 순서로 애트리뷰트를 선언한다.

```
<textarea cols="30" rows="5" (id=" ") (title=" ")></textarea>
```

4.3 폼 엘리먼트

G. <button>

type 애트리뷰트를 선언한다.

- type 애트리뷰트를 button으로 선언하여 열기/닫기, 접기/펼치기 등의 UI를 제어한다.
- 폰 전송 역할을 하는 버튼은 submit 타입을 사용한다.

```
<button type="button">열기</button>
<button type="submit">전송</button>
```

4.4 표 엘리먼트

표 엘리먼트를 아래와 같이 배치한다.

```
<table id="mobTable" class="tblStyle02" summary="시/분,10분,20분,30분,40분,50분,60분">
<caption class="hide">예약시간선택</caption>
<colgroup>
    <col><col><col><col><col>
    <col>
</colgroup>
<thead>
    <tr>
        <th scope="col">시/분</th>
        <th scope="col">10분</th>
        <th scope="col">20분</th>
        <th scope="col">30분</th>
        <th scope="col">40분</th>
        <th scope="col">50분</th>
        <th scope="col">60분</th>
    </tr>
</thead>
<tbody>
    <tr>
        <th scope="row">09시</th>
        <td title="09시00분~09시10분" class="end_time"></td>
        <td title="09시10분~09시20분" class="end_time"></td>
        <td title="09시20분~09시30분" class="end_time"></td>
        <td title="09시30분~09시40분" class="end_time"></td>
        <td title="09시40분~09시50분" class="end_time"></td>
        <td title="09시50분~10시00분" class="end_time"></td>
    </tr>
</tbody>
</table>
```

4.4 표 엘리먼트

A. <table>

일반적으로 레이아웃을 표현하기 위해 사용하지 않고, 2차원의 격자형 데이터를 표현하기 위해 사용한다.
표의 요약 내용을 표기해야 할 때 summary 애트리뷰트를 선택적으로 할 수 있다.

```
<table summary="summary context here">
```

단, Table이 레이아웃으로 사용된 경우, <caption>, <th>, <thead> 등을 사용하지 않아도 되며, <table> 엘리먼트에 tb_layout 클래스를 반드시 선언한다.

```
<table class="tb_layout">
```

B. <caption>

표의 제목을 표현하기 위해 사용한다. caption 엘리먼트는 반드시 선언한다

```
<caption>예약시간선택</caption>
```

C. <colgroup>

<col> 엘리먼트를 그루핑하여 디자인을 제어할 때 선언한다. 이 엘리먼트는 선택적으로 사용한다.<col> 엘리먼트는 5개씩 연속으로 쓰며, 이 후 추가되는 <col> 엘리먼트는 줄바꿈 해준다.

```
<colgroup>
    <col width="100" ><col><col><col><col>
    <col>
</colgroup>
```

4.4 표 엘리먼트

D. <col>

표 각 열의 너비 지정을 위해 선언한다.

width, span 애트리뷰트를 아래와 같은 순서로 선언한다.

```
<col width="100" span="2">
<col width="100"><col width="100">
<col width="100"><col>
<col width="50%"><col width="50%">
```

width, span 애트리뷰트는 필요에 따라 선택적으로 사용한다.

E. <thead>

표 머리글을 그루핑할 때 선언한다. 이 엘리먼트는 th 엘리먼트만으로 그루핑되어있을 때 선언한다.

```
<thead>
    <tr>
        <th scope="col">시/분</th>
        <th scope="col">10분</th>
        <th scope="col">20분</th>
        <th scope="col">30분</th>
    </tr>
</thead>
```

F. <tfoot>

표 바닥글을 그루핑할 때 선언한다. tfoot 엘리먼트는 thead와 tbody 엘리먼트 사이에 위치해야 한다. 이 엘리먼트는 선택적으로 사용한다.

```
<thead> </thead>
<tfoot>
  <tr>
    <th>계</th>
    <td>6,500</td>
    <td>650</td>
  </tr>
</tfoot>
<tbody> </tbody>
```

4.4 표 엘리먼트

G. <th>

scope, abbr, id 순서로 애트리뷰트를 선언한다.

- 표에 셀 제목이 명시되지 않은 경우에도 <th> 엘리먼트를 필수로 선언하여 의미에 맞는 제목을 명시한다. (화면에 표시되지 않도록 CSS로 숨김 처리).
- scope 애트리뷰트는 반드시 선언해야 한다.
- abbr 애트리뷰트는 선택적으로 사용한다.
- id 애트리뷰트는 선택적으로 사용한다.

참고

scope 애트리뷰트: <th> 엘리먼트에 동반되는 애트리뷰트로서 현재의 셀이 어느 셀의 제목인지 범위를 명시한다. scope 애트리뷰트의 값으로는 col, colgroup, row, rowgroup이 있다.

abbr 애트리뷰트: <th> 엘리먼트에 동반되는 애트리뷰트로서 포함하고 있는 콘텐츠를 축약하여 표기할 수 있을 때 약어를 표기하는 데 사용한다. 헤딩 셀의 내용을 반복해서 음성으로 출력하는 도구들은 abbr 애트리뷰트에 표기된 약어를 읽는다.

H. <tbody>

표 본문을 그루핑하기 위해 선언한다. 테이블의 본문(body)이 하나이고 <thead>나 <tfoot>이 없을 경우 생략할 수 있다.

```
<tbody>
    <tr>
        <th scope="row">09시</th>
        <td title="09시00분~09시10분" class="end_time"></td>
        <td title="09시10분~09시20분" class="end_time"></td>
        <td title="09시20분~09시30분" class="end_time"></td>
        <td title="09시30분~09시40분" class="end_time"></td>
        <td title="09시40분~09시50분" class="end_time"></td>
        <td title="09시50분~10시00분" class="end_time"></td>
    </tr>
</tbody>
```

4.5 기타 엘리먼트

A. <a>

href, target, title 순서로 애트리뷰트를 선언한다.

- 새 창으로 페이지를 표시해야 할 때 target 애트리뷰트를 선택적으로 사용한다.
- title 애트리뷰트는 예고 없이 새 창을 표시해야 하거나 이동 경로를 정확히 알 수 없을 때, 또는 브라우저에 독립적으로 툴팁을 표현하

```
<a href="print.html" target="_blank" title="새창">인쇄하기</a>
```

참고

HTML5 에서의 <a>

- HTML5에서의 <a> 엘리먼트 안에 블록 속성의 엘리먼트를 포함할 수 있다.
- href 애트리뷰트 없이 단독으로 <a>OOO 사용이 가능하다.

B. <iframe>

<iframe>은 페이지 성능에 영향을 주기 때문에 가급적 사용하지 않는다.

- 부득이하게 사용해야 할 경우 src, width, height, title, frameborder, marginwidth, marginheight, scrolling 순서로 애트리뷰트를 선언한다.
- 스크린 리더 사용자를 위해 title 애트리뷰트에 제목을 표기한다. 상단에 iframe의 heading 엘리먼트가 있는 경우더라도 반드시 title을

```
<iframe src="nbox.html" width="410" height="800" title="공지사항 게시판" frameborder="0" marginwidth="0" marginheight="0" scrolling="no"></iframe>
```

- Iframe의 내용이 빈 경우더라도 빈 아이프레임라는 것을 사용자에게 알려주도록 한다.

HTML5 DTD 선언 시 frameborder, marginwidth, marginheight, scrolling 애트리뷰트는 사용할 수 없다. 단, 다른 방법으로 구현이 불가한 frameborder 애트리뷰트는 허용한다.

```
<iframe src="nbox.html" width="410" height="800" title="공지사항 게시판" frameborder="0" ></iframe>
```

4.5 기타 엘리먼트

참고

iframe 사용 시 다음과 같은 문제점이 있다.

- iframe을 포함하는 페이지의 도메인과 iframe에서 불러오는 페이지의 도메인이 다르면, 크로스 도메인 설정을 위해 별도의 소스 코드가 추가되어 성능에 영향을 줄 수 있다.
- iframe의 높이값이 콘텐츠에 따라 유동적이어야 한다면 별도의 자바스크립트 코드가 추가되어 성능에 영향을 줄 수 있다.
- 검색 엔진에서 iframe의 내용만 추출하여 표시하면 전체 페이지의 레이아웃이 비정상적으로 보일 수 있으며, 주변 맥락(머리글, 바닥글, 메뉴)이 노출되지 않아 검색 엔진 최적화(SEO) 관점에서 적합하지 않다.
- iframe은 가장 마지막으로 로딩되기 때문에 페이지 로딩 초기에는 화면이 비어 보일 수 있다.
- 모바일에서는 iframe 스크롤에 대한 렌더링이 브라우저별로 다르며, CSS 말 줄임이 동작하지 않는 브라우저가 있다.
- 접근성 보장을 위해 유관부서와 협의 가능 여부를 판단하여 아래 안 중 하나를 선택할 수 있다.
- <iframe>iframe 미지원 장치에서는 내용을 확인할 수 없습니다.</iframe>
- <iframe></iframe>

C. <img>

src, width, height, title, alt, usemap 순서로 애트리뷰트를 선언한다.

- 이미지 내용과 동일한 값을 alt 애트리뷰트에 표기하여, 이미지를 볼 수 없는 환경(스크린 리더, 이미지 서버 장애, 이미지 표시 하지 않음 설정)에서도 내용을 확일할 수 있게 한다.
- title 애트리뷰트를 선언할 경우에도 alt 애트리뷰트를 함께 선언한다.
- title 애트리뷰트는 alt 애트리뷰트값을 축약하거나 브라우저에 독립적으로 툴팁을 표현하기 위해 사용한다.

```
<img src="logo.gif" width="30" height="10" title="고객센터" alt="고객센터, 모듞 궁금증이 해결되는 곳" usemap="#help">
```

4.5 기타 엘리먼트

D. <map>

<map> 엘리먼트의 name 애트리뷰트를 선언하여 <img> 엘리먼트의 usemap 애트리뷰트와 같은 이름으로 커플링한다.

```
<img src="img_main.gif" width="696" height="196" title="오픈이벤트" alt="오픈이벤트, 쿠폰을나눕니다" usemap="#help">
<map name="help">
<area shape="rect" coords="506,48,608,139" href="#" target="_blank" title="고객센터" alt="고객센터, 모든 궁금증이 해결되는 곳">
</map>
```

E. <area>

shape, coords, href, target, title, alt 순서로 애트리뷰트를 선언한다.

- title 애트리뷰트를 선언한 경우에도 alt 애트리뷰트를 함께 선언한다.
- target 애트리뷰트는 새 창으로 페이지를 표시해야 할 때 사용한다.
- title 애트리뷰트는 예고 없이 새 창을 표시해야 하거나 이동 경로를 정확히 알 수 없을 때, alt 애트리뷰트값을 축약하거나, 브라우저에 독립적으로 툴팁을 표현하기 위해 사용한다.

```
<area shape="rect" coords="506,48,608,139" href="#" target="_blank" title="새창" alt="고객센터, 모든 궁금증이 해결되는 곳">
```

5. CSS 코드 작성 규칙

이 장에서는 CSS 코드의 기본 작성 규칙과 인코딩, 선택자, 속성 등의 작성 규칙을 설명한다.

5.1 기본 규칙

A. W3C Validation

모바일에서 CSS는 사용 가능한 Hack과 CSS3 속성을 제외하고 W3C Validation을 통과해야 한다.

B. 영문 소문자 사용

모든 속성은 영문 소문자로만 작성한다.

표 5-1 소문자 작성 예

잘못된 예	올바른 예
.class{Font-Family:AppleGothic}	.class{font-family:AppleGothic}

C. 따옴표 사용 범위

공백이 포함된 폰트명, 한글 폰트명, 문자열 데이터 타입, filter 속성의 파라미터값은 작은따옴표(' ')로 감싸며, @charset 선언 시에는 속성값을 큰따옴표(" ")로 감싼다. 그 외의 경우에는 따옴표를 사용하지 않는다. filter 속성의 파라미터값에 따옴표를 사용하는 것은 선택사항이다. url 데이터 타입에는 작은 따옴표를 사용하지 않는다.

표 5-2 따옴표 사용 예

잘못된 예	올바른 예
font-family:돋움	font-family: '돋움'
filter:progid:DXImageTransform.Microsoft.AlphaImageLoader(src="bg.png", sizingMethod="scale")	filter:progid:DXImageTransform.Microsoft.AlphaImageLoader(src='bg.png', sizingMethod='scale')
background:url(bg.gif) no-repeat	background:url('bg.gif') no-repeat
content: "Chapter: "	content: 'Chapter: '

5.2 들여쓰기

CSS 코드를 작성할 때는 들여쓰기를 하지 않는다.

표 5-3 들여쓰기 사용 예

잘못된 예	올바른 예
#content{border:1px solid #000} #content .class{color:#000}	#content{border:1px solid #000} #content .class{color:#000}

5.3 공백

A. 선택자 간 공백 제거

쉼표로 구분되는 선택자 간 공백은 제거한다.

표 5-4 선택자 간 공백 제거 예

잘못된 예	올바른 예
a:hover,^a:active,^a:focus{text-decoration:underline}	a:hover,a:active,a:focus{text-decoration:underline}

B. 속성 간 공백 제거

속성 간 공백 및 속성 값 사이 공백은 제거한다.

표 5-5 선택자 간 공백 제거 예

잘못된 예	올바른 예
.class p{color:#000;^line-height:18px}	.class p{color:#000;line-height:18px}
font-family:'돋움',^dotum;	font-family:'돋움',dotum;

C. 중괄호 좌우 공백 제거

중괄호 좌, 우 공백은 제거한다.

표 5-6 중괄호 좌, 우 공백 제거 예

잘못된 예	올바른 예
.class p^{color:#000}	.class p{color:#000}
.class p{^color:#000; line-height:18px^}	.class p{color:#000;line-height:18px}

5.4 빈 줄

의미 있는 객체를 구분하기 위하여 코드 그룹 간 1줄씩 빈 줄을 넣을 수 있다. 단, 빈 줄의 간격은 1줄을 초과하지 않게 한다. 빈 줄의 사용은 선택 사항이다.

표 5-7 빈 줄 사용 예

잘못된 예	올바른 예
/* 의미 있는 그룹 1 */ .class1{border:1px solid #000} .class1 img{border:1px solid #000} 빈 줄 빈 줄 /* 의미 있는 그룹 2 */ .class2{border:1px solid #000} .class2 img{border:1px solid #000}	/* 의미 있는 그룹 1 */ .class1{border:1px solid #000} .class1 img{border:1px solid #000} 빈 줄 /* 의미 있는 그룹 2 */ .class2{border:1px solid #000} .class2 img{border:1px solid #000}

5.5 줄 바꿈

선택자, 속성, 속성 값 사이 줄 바꿈은 허용하지 않는다.

표 5-8 줄 바꿈 사용 예

잘못된 예	올바른 예

.class1,
.class2,
.class3{
width:
100px;
color:
#000
}
	.class1,.class2,.class3{width:100px;color:#000}

5.6 인코딩

폰트 이름이 영문이 아닐 때 이를 브라우저에서 바르게 표현하고, HTML에서 불러온 스타일을 제대로 렌더링하려면 반드시 CSS 인코딩을 선언해야 한다. HTML과 동일한 인코딩을 문서 첫 줄에 공백 없이 선언한다. 작성 예는 다음과 같다.

```
@charset "utf-8";
```

파일을 저장할 때는 반드시 선언한 인코딩과 동일한 인코딩을 선택한다.

5.7 선택자

A. 공통 선택자 사용 지양

공통 선택자 '*'는 웹 페이지의 성능을 떨어뜨리고, Internet Explorer에서는 주석까지 영향을 받을 수 있으므로 사용하지 않는다.

B. id 선택자 범위

id 선택자는 2.2 id 및 class 네이밍 규칙에서 정의한 레이아웃(wrap, header, container, content, footer)에만 사용한다.

C. 교차속성 사용

인터넷 익스플로러 6.0 브라우저 환경에서 교차속성 오류가 발생하므로 id와 class, class와 class 간의 교차속성을 사용하지 않는다. 모바일에서는 IE6의 제약을 받지 않으므로 사용할 수 있다.

표 5-9 교차속성 사용 예

잘못된 예	올바른 예
.class{width:980px} .class.bg{background:yellow} .class.bg01{background:green}	.class{width:980px} .bg{background:yellow} .bg01{background:green}

참고

교차속성이란 두 선택자의 조합에 의해 속성이 부여되는 경우를 의미한다.
아래 예에서 background:yellow 속성은 .class와 .bg가 동시에 선언될 때만 의미가 있으며, background:green 속성은 .class와 .bgv1이 동시에 선언될 때만 의미가 있다.

```
.class{width:980px}
.class.bg{background:yellow}
.class.bg01{background:green}
```

인터넷 익스플로러 6.0에서는 id와 class 간 교차속성을 사용할 경우 두 번째 선언된 교차속성은 무시하는 버그가 있으며, class와 class 간의 교차속성을 잘못 해석하는 버그가 있다. 이와 같은 버그를 방지하기 위해 교차속성을 사용하지 않도록 한다.

교차속성을 사용하지 않으면 코드양을 줄이고 각 class 간 독립성을 유지할 수 있다.

5.8 속성

5.8.1 속성 선언 순서(1/2)

속성을 선언할 때는 그 쓰임새가 레이아웃과 관련이 큰 것에서 시작하여 레이아웃과 무관한 것 순서로 선언한다. 관련 속성은 대표되는 속성 다음으로 선언하며, 표 5-11에 표기된 순서대로 선언한다.

대표되는 속성 중 (그룹) 표기된 것은 약식속성으로, 약식속성을 선언했을 때 속성 값의 순서와 동일하게 전체속성을 선언하며 표 5-12에 표기된 순서를 참고한다.

표 5-10 속성 선언 순서

순서	의미	대표되는 속성(그룹)	관련속성
1	표시	display	visibility
2	넘침	Overflow	-
3	흐름	float	clear
4	위치	position	top, right, bottom, left, z-index
5	크기	width & height	-
6	간격	margin & padding (그룹)	-
7	테두리	border (그룹)	-
8	배경	background (그룹)	-
9	폰트	font (그룹)	color, letter-spacing, text-align, text-decoration, text-indent, vertical-align, white-space 등
10	동작	animation	animation, transform, transition, marquee 등
11	기타	-	위에 언급되지 않은 나머지 속성들로 폰트의 관련 속성 이후에 선언하며, 기타 속성 내의 선언 순서는 무관함

* [속성 선언 순서 기준: 1~6: 레이아웃, 7~8: 테두리/배경, 9: 폰트, 10: 동작, 11: 기타]

5.8 속성

5.8.1 속성 선언 순서(2/2)

표 5-11 약식속성의 전체속성 선언 순서

약식속성	전체속성 선언 순서
margin	margin-top, margin-right, margin-bottom, margin-left
Padding	border-top, border-right, border-bottom, border-left, border-width, border-top-width, border-right-width, border-bottom-width, border-left-width, border-style, border-top-style, border-right-style, border-bottom-style, border-left-style, border-color, border-top-color, border-right-color, border-bottom-color, border-left-color, border-image, border-radius, border-collapse, border-spacing
background	background-color, background-image, background-position, background-repeat, background-size, background-attachment, background-origin, background-clip
Animation	animation-name, animation-duration, animation-timing-function, animation-delay, animation-iteration-count, animation-direction
Transition	transition-property, transition-duration, transition-timing-function, transition-delay
font	font-family, font-style, font-size, font-weight, font-variant, line-height

5.8 속성

5.8.2 속성 값 축약

CSS 최적화를 위해 다음과 같이 속성 값을 축약한다.

표 5-12 속성 값 축약 예

축약 전	축약후	설명
#555555	#555	16진수 컬러 코드값
#ff4400	#f40	동일한 값으로 이루어진 16진수 컬러 코드값은 세 자리수로 축약하여 사용한다. 단 인터넷 익스플로러 전용 속성인 CSS filter를 사용했을 경우 축약 속성을 인식하지 못하는 오류가 있기 때문에 속성 값을 축약하지 않는다.
background- position:left center	background-position:0 50%	위치값 문자로 표현한 위치값은 숫자로 변경한다.
top:0px	top:0	0의 단위 속성 값이 0일 경우 단위를 표시하지 않는다.
padding:20px 20px 20px 20px	padding:20px	동일한 속성 값 상, 우, 하, 좌의 속성 값이 동일하면 축약한다.
margin:0 auto 0 auto	margin:0 auto	
padding:20px 30px 50px 30px	padding:20px 30px	

참고
문자로 표현된 위치값을 숫자로 변환할 때 다음과 같이 한다.
top, left → 0 right, bottom → 100%

5.8 속성

3. 약식 속성 사용 범위(1/2)

border 와 background 는 약식 속성을 우선적으로 사용하며, font 약식 속성은 사용하지 않는다.

A. border

속성 값은 border-width, border-style, border-color 순서로 선언한다.
테두리 스타일 속성을 초기 선언할 때는 반드시 border 단일 속성을 사용하고, 이후 테두리의 부분적인 속성이 변경될 경우 border 관련 속성(border-width, border-style, border-color)을 선언한다.
작성 예는 다음과 같다.

```
.class{border:1px solid #ccc}
.class01{border-color:#666}
.class0102{border-style:dotted}
.class_top01 {border-top:1px solid #ccc}
.class_top0201{border-top-color:#666}
.class_top0202{border-top-style:dotted}
```

테두리의 상, 우, 하, 좌 스타일이 2개 이상 다르면, 공통 스타일을 약식 속성으로 선언한 후 다른 부분은 관련 속성으로 선언한다.

표 5-13 테두리 스타일이 2개 이상 다를 경우 약식 속성 선언의 예

잘못된 예	올바른 예
. class{border:1px solid #ddd;border-bottom:1px solid #eee;border-left:1px solid #eee}	.class{border:1px solid;border-color:#ddd #ddd #eee #eee}
.class{border-top:1px solid #ddd;border-right:1px d otted #ddd;border-bottom:1px solid #eee;border-left:1px dotted #eee}	.class{border:1px;border-style:solid dotted;border-color:#ddd #ddd #eee #eee}

5.8 속성

5.8.3 약식 속성 사용 범위(2/2)

B. background

속성 값은 background-color, background-image, background-repeat, background-position, background-size, background-attachment, background-orition, background-clip 순서로 선언한다.

배경 스타일 속성을 초기 선언할 때는 반드시 background 단일 속성을 사용하며, 이후 배경의 부분적인 속성이 변경될 경우 background 관련 속성(background-color, background-image, background-repeat, background-position, background-size, background-attachment, background-orition, background-clip)을 선언한다.

작성 예는 다음과 같다.

```
.class{background:#ccc url(bg.gif) no-repeat}
.class_v1{background-position:0 -50px}
.class_v2{background-position:0 -100px}
```

C. font

폰트 스타일은 약식 속성으로 선언하지 않는다.

폰트 스타일을 약식 속성으로 선언하면 다음과 같은 문제가 발생한다.

아래와 같이 선언하면, font-weight:bold는 상속되지 않고 font 속성의 디폴트값인 font-weight:normal로 변경되기 때문에 불필요한 속성 값을 선언해야 하는 문제가 발생한다.

```
.class{font-weight:bold;font-size:12px;font-family:dotum}
.class p{font:15px gulim}
```

결국, class p의 폰트 스타일은 아래와 같아진다.

```
.class p{font-family:gulim;font-style:normal;font-variant:normal;font-weight:normal;font-size:15px;line-height:normal}
```

5.8 속성

5.8.4 한글 폰트 선언

영문폰트만 선언할 경우 특정 브라우저에서 폰트를 올바르게 출력하지 못하는 경우가 있으므로 한글 폰트 선언시 한글, 영문 폰트를 모두 선언한다.

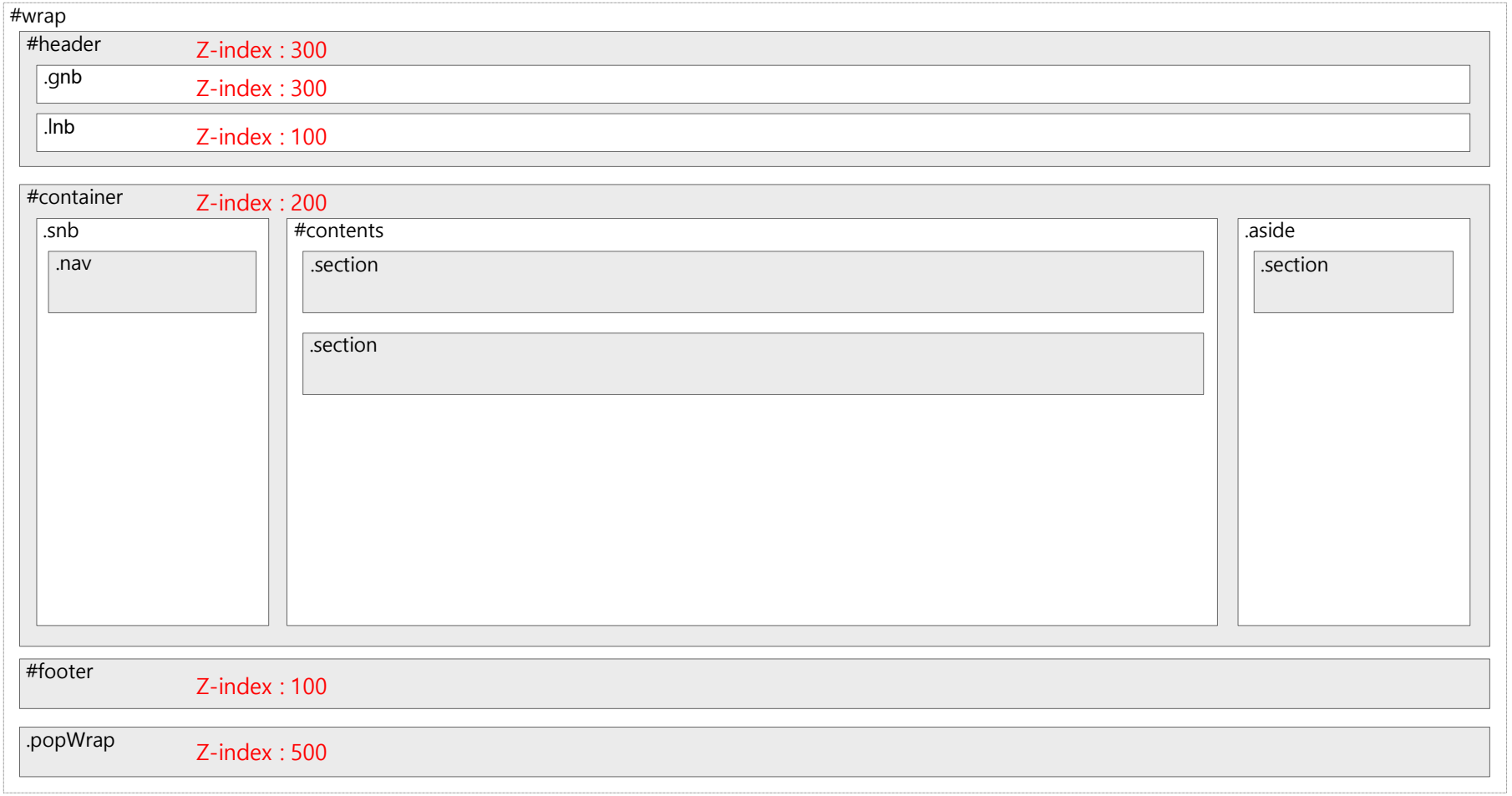
표 5-14 폰트 선언 예

잘못된 예	올바른 예
font-family:'돋움'	font-family:'돋움',dotum

5.9 z-index

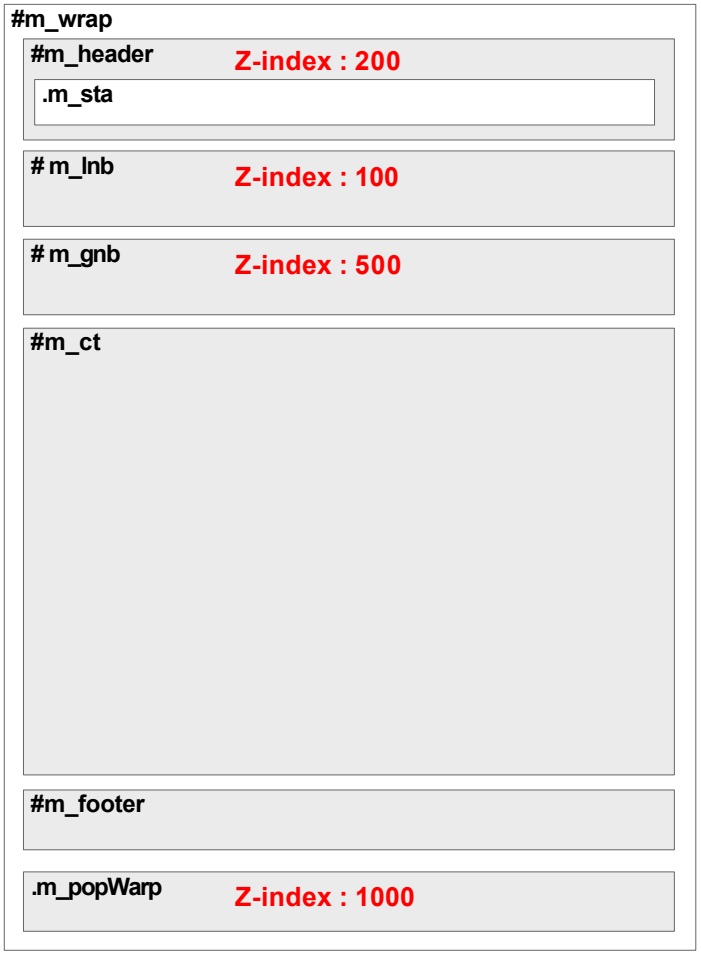
z-index 속성의 속성 값의 범위는 최소 10, 최고 10000이며, 100 단위로 증감한다. 단, 100 단위 사이의 예외 변수가 발생하면 10단위 값을 지정할 수 있다. 다음은 기본 가이드로, 각 서비스의 z-index값은 상황에 맞게 선택적으로 선언한다.

그림 5-1 z-index 권장 선언값



5.9 z-index

그림 5-2 모바일 z-index 권장 선언값



5.10 핵

핵(hack)은 가능한 한 사용하지 않는다. 불가피하게 사용해야 할 때는 그림 5-3에 제시된 핵만 사용하는 것을 원칙으로 한다.

그림 5-3 사용 가능한 핵

<table>
<tr><th>브라우저</th><th colspan="3">표준 DTD</th><th colspan="3">Quirks Mode</th></tr>
<tr><td>IE6</td><td rowspan="3">S{*P:V}</td><td>S{_P:V}</td><td></td><td rowspan="3">S{_P:V}</td><td></td><td rowspan="3"></td></tr>
<tr><td>IE7</td><td>S,x:-moz-any-link, x:default {P:V}</td><td>*+html S{P:V}</td><td rowspan="3">S,x:-moz-any-link, x:default{P:V}</td></tr>
<tr><td>IE8</td><td></td><td></td></tr>
<tr><td>IE9</td><td rowspan="4">S{P:V₩0}</td><td>:root S{P:V₩0 !important}</td><td rowspan="4">:root S{P:V}</td><td rowspan="3"></td><td rowspan="4">:root S{P:V}</td></tr>
<tr><td>Firefox</td><td rowspan="3">S,x:-moz-any-link, x:default{P:V}</td><td rowspan="3"></td></tr>
<tr><td>Safari/ Chrome</td><td rowspan="2"></td><td rowspan="2"></td></tr>
<tr><td>Opera</td><td></td><td></td></tr>
</table>

[약어 표기 – S(Selector): 선택자, P(Property): 속성, V(Value): 속성 값]

5.10 핵

모바일에서도 핵 은 가능한 한 사용하지 않는다. 사파리, 안드로이드, 돌핀, 오페라 모바일은 class로 분기하여 대응하므로 CSS3를 위한 속성 이외에는 핵을 사용하지 않는다. 이외의 브라우저에서 불가피하게 사용해야 할 때는 그림 5-4에 제시된 핵만 사용하는 것을 원칙으로 한다.

그림 5-4 모바일에서 사용 가능한 핵

브라우저	CSS 파일명	CSS3 대응	IE 엔진을 사용하는 경우
사파리, 안드로이드, 돌핀, 오페라 모바일	w.css	S{P3:V}	
기타(default)	e.css		표 5-16 참조

[약어 표기 – S(Selector): 선택자, P(Property): 속성, V(Value): 속성 값, P3(CSS3 Property): CSS3 속성]

5.11 미디어 타입

미디어 타입 선언을 위한 명령문(중괄호 포함)은 다음과 같이 세부 스타일을 지정하는 명령문과 줄로 구분한다.
인쇄를 위한 미디어 타입은 기본 CSS 파일의 가장 아랫부분에 선언하며 별도의 CSS 생성은 허용하지 않는다.

```
...
...
/* For Print */
@media print{
#header,.snb,.aside{display:none}
}
```

참고
미디어 타입은 각종 미디어(프린터, 모바일, 보조공학기기 등)에 대응하는 별도의 CSS 코드를 작성할 수 있게 한다. CSS 2.1 Specification에 따라 대응 가능한 미디어 타입은 아래와 같으며, 가장 범용적으로 사용하는 타입은 print 타입이다.

- all (모든 출력 장치)
- aural (음성 출력)
- braille (점자 출력)
- handheld (포켓, 모바일)
- print (인쇄)
- projection (투사 장치)

5.12 CSS 선언 타입

CSS 선언 타입은 크게 세 가지로 분류하며, 용도에 맞게 사용한다.

A. External type

CSS를 선언하는 가장 기본적인 방식으로, CSS 파일이 별도로 존재하는 형태이다. link 엘리먼트를 통해 HTML과 CSS 파일을 연결한다.
작성 예는 다음과 같다.

```
<link rel="stylesheet" type="text/css" href="../css/common.css">
```

B. Internal(embedded) type

HTML 파일의 head 앞에 스타일을 선언하는 방식으로, 단발성 페이지의 CSS 분량이 작을 경우 사용한다.
작성 예는 다음과 같다.

```
<head>
...
<style type="text/css">
...
</style>
</head>
```

C. Inline type

HTML의 개별 엘리먼트에 style 속성을 이용하여 스타일을 선언하는 방식으로, 속성 값이 동적으로 변경되어야 하는 경우 사용한다. 속성 값에 사용되는 %와 같은 특수기호는 Character entity references로 변환하지 않는다.
작성 예는 다음과 같다.

```
<div style="top:0;left:50%">
...
</div>
```

5.13 주석

5.13.1 한글 폰트 선언

CSS 주석은 아래와 같이 표기하며, 기본 형식에 맞게 작성한다.

```
/* 주석 내용 */
```

- 주석 기호와 주석 내용 사이에는 반드시 공백 한 칸이 있어야 한다.
- 주석 기호와 주석 내용 사이의 줄 바꿈은 허용하지 않는다. 단, 주석 내용 간 줄 바꿈은 허용한다.
- 종료 주석은 사용하지 않는다

주의

마크업과 개발의 편의를 위해 작성한 주석은 실제 서비스를 적용할 때 반드시 삭제한다. 단, 작성자 정보는 삭제하지 않는다.

5.13.2 작성자 정보 표기

작성자 표기는 문서의 최상단 1회만 작성하며, 작성자 정보에는 소속 부서, 영문 이름 이니셜, CSS 파일 생성 날짜(YYMMDD 형식)를 작성한다. 유지보수만 담당하는 경우라도 모두 기입한다. 작성자 정보 밑에는 빈 줄(한 줄)을 두어 스타일을 선언하는 문장과 구분되도록 한다. 동일 파일을 유지보수 하는 경우, 작성자 바로 아래 줄에 동일한 형식으로 작성한다.

```
/* mindsLAB CMH 20190911, YMJ 20190918 */
/* mindsLAB CMH 20191030 *
/  빈 줄
```

5.13 주석

5.13.3 의미 있는 그룹 영역의 주석 표기

의미 있는 객체를 구분하기 위한 주석은 영역의 윗부분에 표기한다. 초기화와 레이아웃 스타일 그룹을 제외한 의미 있는 그룹 영역의 주석 표기는 선택 사항이다.

작성 예는 다음과 같다.

```
/* 공통_subMenu */
.my_snb{width:182px}
.my_snb li .num{padding-left:4px;color:#919190;font-size:11px;letter-spacing:0}
.my_snb li.on a,.my_snb li.on .num{color:#259e0b}
.my_snb li a{color:#424242}
```

A. 초기화 스타일 그룹

CSS 초기 파일에 따라 초기화 속성은 /* reset */으로 그루핑한다.

```
/* reset */
html, body, div, span, applet, object, iframe, h1, h2, h3, h4, h5, h6, p, blockquote, pre, a, abbr, acronym, address, big, cite, code, del, dfn, em, img, ins, kbd, q, s, sa
mp, small, strike, strong, sub, sup, tt, var, b, u, i, center, dl, dt, dd, ol, ul, li, fieldset, form, label, legend, table, caption, tbody, tfoot, thead, tr, th, td, article, aside, ca
nvas, details, embed, figure, figcaption, footer, header, hgroup, menu, nav, output, ruby, section, summary, time, mark, audio, video
{margin:0;padding:0;border:0;font-size:100%;vertical-align:baseline;box-sizing:border-box;}
article, aside, details, figcaption, figure, footer, header, hgroup, menu, nav, section {display:block;}
```

B. 레이아웃 스타일 그룹

레이아웃을 위한 스타일 선언 시 /* Layout */으로 그루핑한다.

```
/* Layout */
#wrap{...}  #h
eader{...}  #co
ntainer{...}  #f
ooter{...}
```

5.14 파일 분기

파일은 독립된 한 서비스 내에 반드시 하나의 파일을 생성하며, 분기를 허용하는 경우는 다음과 같다.

- 한 서비스가 사용자 화면/어드민으로 구성되어 전체 레이아웃이 다를 경우
- 웹폰트를 적용할 경우
- 개편 시 개발상의 이슈로 이전에 분리되어 있던 CSS를 합칠 수 없는 경우
- 한 서비스 내에 단발성 페이지로 존재하나 CSS를 임베드하기 어려운 경우
- 반응형웹을 대응할 경우

모바일에서는 해상도의 차이와 브라우저별 렌더링 차이로 단말기 별로 뷰포트, CSS, <body>의 class를 다르게 지정하여 분기한다.

- 파일분기 규칙은 모바일 시장상황에 따라 자주 변화함으로 상황에 맞는 기준으로 적용한다.
- 서비스 출시일에 따라 적용하는 파일 분기 규칙과 다르므로 각 서비스에 맞춰 적용하고, 신규서비스는 최신 파일분기 규칙을 적용한다.

5.15 초기파일

CSS를 새로 작성할 때는 아래 파일을 기본으로 한다. 초기 파일에는 스타일 속성 초기화 문장이 포함되어 있으며, 초기화 문장은 변경할 수 없다. CSS 작성 시 반드시 회사명/팀명/작성자/날짜 를 기입한다.

```
@charset "utf-8";
/* MINDsLab / UX/UI Team / YMJ 20180101 */

/* reset */  body,p,h1,h2,h3,h4,h5,h6,ul,ol,li,dl,dt,dd,table,th,td,form,fieldset,legend,input,textarea,button,select{margin:0;
padding:0}  body,input,textarea,select,button,table{font-family:'돋움',Dotum,AppleGothic,sans-
serif;font-size:12px}  img,field
set{border:0}  ul,ol{list-style:n
one}  em,address{font-style:n
ormal}  a{text-decoration:non
e}
a:hover,a:active,a:focus{text-decoration:underline}
```

5.15 초기파일

모바일에서 초기 파일은 브라우저에 따라 w.css, e.css 파일을 작성한다.

w.css

```
@charset "utf-8";
/* MINDsLab / UX/UI Team / YMJ 20180101 */

/* reset */  body,p,h1,h2,h3,h4,h5,h6,ul,ol,li,dl,dt,dd,table,th,td,form,fieldset,legend,input,textarea,button,select{margin:0;
padding:0}  body,input,textarea,select,button,table{font-size:14px;line-height:1.25em}
body.s,.s input,.s textarea,.s select,.s button,.s table{font-family:Helvetica}  body{position:r
elative;background-color:#f4f4f4;color:#000;-webkit-text-size-adjust:none}  img,fieldset{b
order:0}
ul,ol{list-style:none}  em,address{fo
nt-style:normal}  a{color:#000;text-
decoration:none}  table{border-col
lapse:collapse}  hr{display:none !i
mportant}
.u_hc,.u_skip{visibility:hidden;overflow:hidden;position:absolute;left:-999em;width:0;height:0;font-size:0;line-height:0}
.u_vc{position:absolute;z-index:-1;font-size:1px;line-height:1px;color:transparent}
#m_ct{clear:both;width:100%;background-color:#fff}  #m_ct::after{display:block;cl
ear:both;height:1px;margin-top:-1px;content:''}
```

e.css

```
@charset "utf-8";
/* MINDsLab / UX/UI Team / YMJ 20180101 */

/* reset ~ Footer */
body,input,textarea,select,button,table{font-family:'나눔고딕',NanumGothic,'맑은 고딕','Malgun Gothic','돋움',Dotum,'굴림',Gulim,Helvetica,sans-serif}
.u_vc{*left:-9999em;left:-9999em₩9}
.u_ts{_font-size:1.3em;_line-height:33px}
.u_ts_a{_vertical-align:top}
.u_ts img{_vertical-align:top}
.u_gnbu_w,.u_gnbu,.u_ft,.u_hsft,.u_ftsw{_zoom:1}
.atcpa{_white-space:normal}
```

6. Javascript 코드 작성 규칙

이 장에서는 Javascript 코드의 기본 작성 규칙 및 금지항목 및 주의사항을 설명한다

6.1 기본규칙

A. <script> 태그

자바스크립트를 HTML 페이지에 삽입하는 방법은 <script> 태그를 이용하는 것이다.
<script>에는 아래와 같이 여러 가지 속성이 있지만 보통 "src"와 "type" 정도만 사용하고, 사실 "type"은 생략하는 것이 안전하다.

표 6-1 script 태그 설명

태그명	설명
src	외부에서 불러올 스크립트 파일의 위치를 지정한다
charset	src로 불러온 스크립트 파일의 문자셋을 지정한다. 대게 이 속성값은 브라우저에서 무시되므로 잘 쓰지 않는다.
Language	"javascript", "VBScript" 등 코드 블록에서 사용한 스크립트 언어를 지정할 목적이었지만 대부분 브라우저에서 무시된다. 폐기된 속성이므로 사용하면 안 된다.
Type	language 속성을 대체할 의도로 만들어졌으며 스크립트 언어의 컨텐츠 타입을 지정한다. 잘못 지정하면 스크립트가 무시될 수도 있기때문에 생략할 것을 권장한다. 브라우저 호환성을 위해 "text/javascript"를 쓰기도 하지만, 기본 값이 "text/javascript"이므로 생략해도 안전하다.
defer	외부 스크립트 파일을 다운받을때, 문서의 파싱을 중단시키지 않고 스크립트를 다운받게 한다. 문서 파싱과 동시에 스크립트를 다운받지만 문서 파싱이 끝난 후에 스크립트가 실행된다.
async	defer와 같이 문서 파싱과 동시에 스크립트를 다운받는다. defer와 다른 점은 문서 파싱이 끝나지 않았어도 스크립트 다운로드가 완료되면 바로 실행된다.

6.1 기본규칙

B. 세미콜론

자바스크립트에서 문장은 세미콜론(;)으로 끝나야 하지만, 문법적으로 이를 강제하지 않기 때문에 생략할 수 있다. 자바스크립트 파서가 이를 자동으로 삽입해주기 때문이다. 하지만 이런 세미콜론(;) 자동 삽입은 종종 의도하지 않았던 코드(전혀 다른 코드)로 해석되어 생각하지 못한 오류를 만들고 디버깅을 어렵게 한다. 또한 자바스크립트 파서가 세미콜론(;)의 위치를 계산하고 삽입하는데 추가적인 비용이 발생한다.

표 6-2 세미콜론 사용 예

잘못된 예

```
function parserTest(options) {  cons
    ole.log('test start!!!')  (options || [
    ]).forEach(function(i) {
    ...
    })
} /**
* parser는 아래와 같이 해석하고, 세미콜론을 삽입
* function parserTest(options) {
*     console.log('test start!!!')(options || []).forEach(function(i) {...});
* }
*/
```

올바른 예

```
function parserTest(options) {  cons
    ole.log('test start!!!');  (options || [
    ]).forEach(function(i) {
    ...
    });
}
```

C. 줄 길이

한 라인의 문자수가 80자 이하로 작성한다. 80자 이상일 경우 다음 줄로 넘길때 연산자 다음으로 넘기는것이 좋고 이상적으로는 콤마(,) 다음으로 넘긴다. 한개의 라인이 80자 이상이어서 라인을 내렸을경우 연장되는 라인은 스페이스 4개로 들여쓰기 한다.

```
// Four-space, wrap at 100. Works with very long function names, survives
// renaming without reindenting, low on space.  goog.foo.ba
r.doThingThatIsVeryDifficultToExplain = function(
    veryDescriptiveArgumentNumberOne, veryDescriptiveArgumentTwo,
    tableModelEventHandlerProxy, artichokeDescriptorAdapterIterator) {
  // ...
};
```

6.1 기본규칙

D. 따옴표 사용 예

안에서 HTML 작성 시 큰 따옴표("")가 기본으로 사용되어야 하므로 백슬래시(₩)를 이용하여 일일이 이스케이프(escape) 하는 상황이 발생하면 가독성이 크게 떨어지기 때문에 쌍따옴표(“”)를 지향하고 홑따옴표('')을 사용한다.

표 6-3 따옴표 사용 예

잘못된 예	올바른 예
`var foo = "bar";`	`var foo = 'bar';`

표 6-4 이스케이프 상황 예

쌍따옴표 일때	홑따옴표 일때
`var anchor = "<a href=₩"/" + foo + ".html₩">" + foo + "</a>";`	`var anchor = '<a href="/' + foo + '.html">' + foo + '</a>'; // correct`

E. 숫자

숫자는 10진수 정수, 지수표현법을 이용한 정수, 16진수 정수, 부동 소수점을 이용한 정수만 사용하며, 소수점 앞 뒤에는 숫자가 최소 하나는 있어야 한다.

8진수는 사용하지 않는다.

표 6-5 숫자 표현 예

잘못된 예	올바른 예

`var num = 10.; // 소수점 뒤에 숫자 없음`
`var num = .1; // 소수점 앞에 숫자 없음`
`var num = 010; // 8진수 사용 안함`
	`var num = 10;`
	`var num = 10.0;`
	`var num = 10.00;`
	`var num = 0xB2;`
	`var num = 3e1;`

6.2 들여쓰기

절대 스페이스와 탭을 섞어 쓰지 않는다. 들여쓰기에 대한 규칙은기본 1 Tab = 4 space 편집기의 들여쓰기로 한다. 하지만 프로젝트의 성격에 따라 선택따라, 들여쓰기 규정이 미리 정해져 있을 경우 2문자 또는 4문자를 사용한다. 설정을 규칙대로 설정 해놓고 사용할 것을 권장한다.

표 6-6 들여쓰기 사용 예

잘못된 예	올바른 예

`function test() {  console.log('`
`들여쓰기 테스트');`
`}`
	`function test() {  console.log('들`
	`  여쓰기 테스트');`
	`}`

6.3 연산자

6.3.1 공백

연산자의 목적를 명확하게 보여주기 위해 두 개의 피연산자와 함께 사용하는 연산자는 사용 시 연산자 앞 뒤에 공백을 한 칸씩 추가해주도록 한다.

표 6-7 연산자 삽입시 공백 삽입 위치

잘못된 예	올바른 예
`var isSuccess = (condition<=10);`	`var isSuccess = (condition^<=^10);`

`var i; for (i=0; i<10; i++) {`
`  work();`
`}`
	`var i; for (i ^= ^0; i ^<^10; i++) {`
	`  work () ;`
	`}`

6.3.2 할당 연산자

할당 연산자 사용시 할당 값이 비교 연산식이면 괄호로 감싸준다.

표 6-8 할당 연산자에 대한 괄호 사용 예

잘못된 예	올바른 예
`var isBaby = age < 10;`	`var isBaby = (age < 10);`

6.3.3 동등 연산자

동등 비교를 할 때 타입 강제 변환의 위험이 있으면 무조건 ==, != 대신 ===, !==를 사용해준다.

표 6-9 동등 연산자에 대한 사용 예

잘못된 예	올바른 예
`console.log('1' == 1);`	`console.log('1' === 1);`

6.4 괄호

if/while/do/for 문은 한 줄짜리 블록 일 경우 {}를 생략할 수 있지만 이런 패턴은 코드 구조를 애매하게 만들어 가독성이 떨어지고, 문법적 오류가 아니기 때문에 디버깅이 어렵다. 이런 코드는 이후 오류 발생 확률이 높은 잠재적 위험 요소가 되므로 반드시 중괄호로 블록을 명확하게 만든다.

표 6-10 소괄호 사용 예

잘못된 예	올바른 예
`// 앞 뒤에 불필요한 공백 들어감` `var isBaby = ( age <= 10 );`	`var isBaby = (age <= 10);`

`// 앞 뒤에 불필요한 공백 들어감`
`var i;`
`for (i = 0; i < 10; i++) {`
`  work( hour, amount );`
`}`
	`var i;`
	`for (i = 0; i < 10; i++) {`
	`  work(hour, amount);`
	`}`

표 6-11 중괄호 사용 예

잘못된 예	올바른 예

`if (true)`
`  doSomething();`
`else`
`  doNotAnything();`
	`if (true) {`
	`  doSomething();`
	`} else {`
	`  doNotAnything();`
	`}`

`// 객체 리터럴 규칙을 지키지 않음  getDe`
`partment({name: "홍길동", age: 26});`
	`getDepartment({`
	`  name: "홍길동",`
	`  age: 26`
	`});`

6.5 주석

A. 한 줄 주석

- 하단의 코드를 설명하기 위해 독립된 줄에 주석 작성한다.
- 주석 앞의 코드를 설명하기 위해 줄 끝에 주석 작성한다.
- 여러 줄에 걸친 설명일 때는 사용하지 않는다.
- 독립된 줄에 한 줄 주석 작성 시에는 설명할 코드와 같은 단계 들여쓰기 사용하며, 이전 줄은 한 줄 비운다.

표 6-12 한줄 주석 사용 예

잘못된 예	올바른 예

`if (age > 20) {`
`  // 이 줄은 나이가 20살 이상일 때만 실행됩니다.`
`  showAdultMovie();`
`}`
	`if (age > 20) {`
	
	`  // 이 줄은 나이가 20살 이상일 때만 실행됩니다.`
	`  showAdultMovie();`
	`}`

`if (age > 20) {`
 	
`  // 이 코드는 만으로 나이가 20세 이상인 것이 아니고`
`  // 그냥 20세 이상 아니아니 이게 아니고`
`  // 아 아닙니다 만으로 나이가 20세 이상인`
`  // 사람을 판별하기 위한 조건문입니다`
`  showAdultMovie();`
`}`	

	`// 코드블럭 주석은 한 줄 주석 사용`
	`...`
	`// var foo = '';`
	`// var bar = '';`
	`// var quux;`

6.5 주석

B. 여러 줄 주석

- 첫 번째 줄은 여러 줄 주석을 여는 방법인 /*를 삽입하며, 다른 텍스트는 넣지 않는다.
- 그 다음 줄 부터는 *와 텍스트를 입력하는데 첫 째 줄의 *과 열을 맞춘다. *과 텍스트 사이에는 공백 하나를 넣어준다.
- 마지막 줄은 여러 줄 주석을 닫는 방법인 */를 삽입하며 다른 텍스트는 넣지 않는다.

표 6-13 여러 줄 주석 사용 예

잘못된 예

```
if (age > 20) {
    / *
      *이 코드는 만으로 나이가 20세 이상인 것이 아니고
      *그냥 20세 이상 아니아니 이게 아니고
      *아 아닙니다 만으로 나이가 20세 이상인
      *사람을 판별하기 위한 조건문입니다.
      */  showAdultMo
    vie();
}
```

```
if (age > 20) {
    // 이 코드는 만으로 나이가 20세 이상인 것이 아니고
    //그냥 20세 이상 아니아니 이게 아니고
    //아 아닙니다 만으로 나이가 20세 이상인
    //사람을 판별하기 위한 조건문입니다
    showAdultMovie();
}
```

올바른 예

```
if (age > 20) {

    / *
      * 이 코드는 만으로 나이가 20세 이상인 것이 아니고
      * 그냥 20세 이상 아니아니 이게 아니고
      * 아 아닙니다 만으로 나이가 20세 이상인
      * 사람을 판별하기 위한 조건문입니다.
      */  showAdultMo
    vie();
}
```

6.6 변수 선언

A. 변수 선언 방법

변수 연산자는 반드시 사용 전에 선언되어야 한다. 가독성을 위해 한 개의 var을 이용해 함수의 맨 윗줄에 선언하며, 대입 연산자는 같은 단계의 들여쓰기를 사용한다. 마지막 변수 선언 전까지는 ,로 구분을 해주며 마지막 변수에는 세미콜론(;)을 붙여준다. 초기화 된 변수는 반드시 초기화 되지 않은 변수보다 앞에 위치 시킨다.

표 6-14 변수 선언 예

잘못된 예

```
// 대입연산자의 들여쓰기가 맞지 않음
var name    = "홍길동",
      age = 23,
      home    = "방화동";
```

```
// 들여쓰기가 맞지 않음
var name = "홍길동",
age = 23,
home = "방화동";
```

```
// 한 줄에 여러 변수 선언
var name    = "홍길동", age = 23, home      = "방화동";
```

```
// 초기화 되지 않은 변수가 먼저 위치함
var name ,
      age = 23,
      home    = "홍길동";
```

올바른 예

```
var name = "홍길동",
      age    = 23,  ho
      me = "방화동";
```

6.6 변수 선언

B. 변수명 규칙

변수명은 낙타표기법을 사용한다. $나 ₩는 사용하지 않으며 _도 사용을 자제한다. 변수명은 소문자로 시작하며 새로운 단어의 첫 번째 문자는 대문자를 사용한다. 변수명의 첫 단어는 반드시 명사를 사용해주어 함수와 구분하기 쉽도록 한다.

표 6-15 변수명 규칙 예

잘못된 예	올바른 예
`// 대문자로 시작` `var FirstName = "길동";`	`var firstName = "길동";`
`// 빈칸 들어감` `var first_name = "길동";`	
`// 동사로 시작` `var getFirstName = "길동";`	

변수를 값이 변하지 않는 상수로 사용하려면 이름의 모든 문자를 대문자로 사용하며 단어 사이에 밑줄(_)을 넣어준다.

```
var TOTAL_COUNT = 1,  TOTAL_P
      OPULATION = 1000;
```

6.7 함수 선언

A. 함수 선언 방법

객체에 선언된 메서드가 아니라면 반드시 함수 선언 형식을 사용한다. 함수 표현식, new를 이용한 생성자 형식을 사용하지 않는다. (객체 생성 시 사용)함수명과 여는 괄호 '(' 사이에는 공백 넣지 않는다. '(' 와 ')' 사이에는 공백 하나를 넣는다. 인자 입력시 , 뒤에 공백을 넣는다. 함수 본문은 한 단계 들여쓰기를 사용한다.

표 6-16 함수선언 예

잘못된 예

```
// 함수 명과 괄호 사이에 공백
function working^(arg1, arg2) {
    return money;
}
```

```
// 함수 표현식 사용
var working = function(arg1, arg2) {
    return money;
}
```

```
// 생성자 방식 사용
var money = new Function("arg1", "arg2");
```

올바른 예

```
function working(arg1, arg2) {
    return money;
}
```

익명 함수는 객체에 메서드 할당 시나, 다른 함수에 인자로 전달할 때 사용한다.

```
person.name = function() {
    // 코드
};

getName(function() {
    // 코드
});
```

6.7 함수 선언

B. 함수명 규칙

함수명 역시 낙타표기법을 사용한다. 첫 번째 단어는 동사를 사용하여 변수명과 구분해준다.

표 6-17 함수명 규칙 예

잘못된 예	올바른 예

// 대문자로 시작
function GetName() {
 	
};
	function getName() {
	
	};

// 명사 사용
function name() {
 	
};	

// _ 사용
function get_name() {
 	
};	

생성자 함수는 new를 통해 생성할 때 실행되는 함수며 객체 인스턴스를 생성할 때 사용하므로 동사로 시작하지 않고, 첫 문자를 대문자로 시작한다.

표 6-18 생성자 함수명 규칙 예

잘못된 예	올바른 예

// 대문자로 시작
function enemy() {
 	
};
	function Enemy() {
	
	};

6.8 문장 규칙

6.8.1 return문

반환하는 값의 명확한 의미를 알기 위해 괄호를 사용하지 않는다.

표 6-19 return문 작성 예

잘못된 예	올바른 예
`return (age <= 7 ? "baby" : "teenager");`	`var generation = (age <= 7 ? "baby" : "teenager");` `return generation;`

6.8.2 if 문

한 문장이라도 중괄호를 포함한다.

표 6-20 if문 작성 예

잘못된 예	올바른 예

`if (person != null)`
`  work();`
	`if (person != null) {`
	`  work();`
	`}`

6.8 문장 규칙

6.8.3 for 문

변수는 for문의 초기화 구문 이전에 선언한다.

표 6-21 for문 작성 예

잘못된 예	올바른 예

`for (var i = 0; i < 10; i++){`
 	
`}`
	`var i = 0,`
	`len = 10;`
	`for (i = 0; i < len; i++){`
	
	`}`

주의

보통 객체 순회에는 for-in을 사용한다.

자바스크립트에서는 배열(Array)도 객체(Object)이지만 배열을 순회할 때 for-in을 사용해서는 안된다.

for-in은 프로토타입 체인에 있는 모든 프로퍼티를 순회하기 때문에 for를 사용할 때보다 보통 10배 이상 느리고, index 순서대로 배열을 순회한다고 보장할 수 없다.

순회 순서는 브라우저에 따라 다를 수 있다.

표 6-22 배열을 순회할 때의 for문 사용 예

잘못된 예	올바른 예

`var scores = [70, 75, 80, 61, 89, 56, 77, 83, 90, 93, 66];`
`var total = 0;`
`var score;`
 	
`for (score in scores) { tot`
`al += scores[score];`
`}`
	`var scores = [70, 75, 80, 61, 89, 56, 77, 83, 90, 93, 66];`
	`var total = 0;`
	`var i = 0;`
	`var len = scores.length;`
	
	`for (; i < len; i += 1) {`
	`total += scores[i];`
	`}`

6.8 문장 규칙

4. switch 문(1/2)

- case는 switch를 기준으로 한 단계 들여쓰기 한다.
- 첫 번째 case를 제외한 case와 default 사이에는 한 줄의 공백이 있어야 한다.
- default를 제외한 case는 return, break, throw 중 하나로 마쳐야 하고 다음 case로 의도적으로 넘기는 부분은 주석처리를 한다.
- default를 의도적으로 생략했으면 주석처리 한다.

```
switch (num) {
    case 0:
        break;
    case 1:
        break;
    default:
        break;
}
```

```
switch (num) {
    case 0:
        // 다음 case와 같은 동작 처리

    case 1:
        break;
    default:
        break;
}
```

주의

switch는 if에 비해 간결한 코드를 작성할 수 있지만, 비교적 많은 메모리를 사용하고 저버전 IE에서는 처리 성능도 좋지 않다.
최신 버전 IE와 크롬과 같은 고성능 브라우저만 지원하는 서비스라면 이런 부분을 크게 고민하지 않아도 되겠지만, 저버전 IE를 포함한 다양한 환경을 지원해야 하는 서비스라면 switch문의 득과 실에 대해 따져볼 필요가 있다.

- if문은 원하는 조건이 나올 때까지 순차적으로 모든 비교문을 순회하면서 비교
- switch문은 jump-table을 사용하여 한 번에 원하는 곳으로 이동

때문에

- if문은 조건 문의 개수만큼 O(n)의 시간 복잡도를 갖게 되어 성능의 단점
- switch문은 case의 개수만큼 jump-table을 차지하므로 메모리의 단점
  - 저버전 IE에서는 성능에도 문제가 있음
  - case의 조건이 정수형이 아닌 경우(문자열 또는 연산)라면 jump-table을 쓸 수 없어 성능에도 이득 없음

6.8 문장 규칙

6.8.4 switch 문(2/2)

표 6-23 조건문 개수에 대한 if, switch문 작성 예

잘못된 예

```
switch(foo) {
    case 'alpha':
        alpha();
        break;
    case 'beta':
        beta();
        break;
    default:
        ...
        break;
}
```

올바른 예

```
// 객체 리터럴 방식
var switchObj = {
    alpha: function() {
        ...
    },
    beta: function() {
        ...
    },
    _default: function() {
        ...
    } };
switchObj[foo](args);
```

```
// 모듈 방식
var switchModule = (function() {
    return {
        alpha: function() {
            ...
        },
        beta: function() {
            ...
        },
        _default: function() {
            ...
        }
    };
})();
switchModule[foo](args);
```

6.9 금지항목 및 주의사항

6.9.1 순환문에서 continue 사용금지

continue를 사용하면 JavaScript 엔진에서 별도의 실행 컨텍스트를 만들어 관리하므로 성능 문제가 발생할 수 있고, 순환문의 성능이 전체 성능에 많은 영향을 주기 때문에 사용하지 않는 것이 좋다.
또한 continue 문을 잘 사용하면 코드를 간결하게 작성 할 수 있지만, 과용할 경우 디버깅 시 개발자의 의도를 파악하기 어렵게 만들어 유지 보수에 문제가 생길 수 있다.

표 6-24 조건문 작성 예

잘못된 예

```
var loopCount = 0;
var i = 1; for (; i < 10; i += 1){
    if (i >= 5) {
        continue;
    }
    loopCount += 1;
}
```

올바른 예

```
var loopCount = 0;
var i = 1; for (; i < 10; i += 1){
    if (i < 5) {
        loopCount += 1;
    }
}
```

순환문 내부에서 특정 코드의 실행을 건너뛸때는 조건문을 사용한다.

6.9 금지항목 및 주의사항

6.9.2 순환문 안에서 try-catch 사용금지

예외 처리를 위해 try-catch를 사용하는데, catch문이 실행될 때 내부적으로 예외 객체를 변수에 할당하게 된다.
때문에 try-catch가 순환문 안에서 사용될 경우, 순회가 반복될 때마다 런타임의 현재 스코프에서 예외 객체 할당을 위한 새로운 변수가 생성된다.

표 6-25 조건문 작성 예

잘못된 예

```
var i = 0;
var len = array.length;
for (; i < len; i += 1) {
    try {
        // ...
    } catch (error) {
        // ...
    }
}
```

올바른 예

```
var i = 0;
var len = array.length; function doSomething() {
    try {
        ...
    } catch (error) {
        ...
    }
}

for (; i < len; i += 1) {
    doSomething();
}
```

try-catch를 감싼 함수를 만들고, 순환문 내부에서 함수를 호출한다.

6.9 금지항목 및 주의사항

6.9.3 배열의 요소 삭제 시 delete 사용금지

객체의 프로퍼티를 삭제할 때 delete를 사용하는데 단순히 undefined로 설정되는 것이 아니라 프로퍼티 자체가 완전히 삭제되어 더 이상 존재하지 않게 된다. 자바스크립트에서는 배열(Array)도 객체(Object)이지만 배열의 요소 삭제는 객체의 프로퍼티 삭제와 조금 다르게 동작한다.

배열의 요소에 delete를 사용할 때 사실은 배열의 길이가 줄어들거나 삭제된 요소를 기준으로 shift된 효과를 기대하겠지만, 결과적으로는 해당 요소의 위치에 구멍이 생길 뿐 변하는 것은 아무것도 없다. 삭제된 요소 외에 다른 어떤 요소도 영향을 받지 않으며, 배열의 길이 또한 변하지 않는다.

표 6-26 조건문 작성 예

잘못된 예

```
var numbers = ['zero', 'one', 'two', 'three', 'four', 'five']; delete
numbers[2];                    // ['zero', 'one', undefined, 'three', 'four', 'five'];
```

올바른 예

```
var numbers = ['zero', 'one', 'two', 'three', 'four', 'five'];
numbers.splice(2, 1);                  // ['zero', 'one', 'three', 'four', 'five'];
```

배열의 요소를 삭제할 때는 splice 또는 length를 사용한다.

참고

배열의 사이즈를 줄이고 싶다면 length를 사용한다.

```
var numbers = ['zero', 'one', 'two', 'three', 'four', 'five'];
numbers.length = 4;                    // ['zero', 'one', 'two', 'three'];
```

6.9 금지항목 및 주의사항

6.9.4 두 번 이상 사용되는 DOM 요소

DOM 요소에만 국한되는 내용은 아니지만 DOM 요소에서 더 두드러지게 성능 저하가 나타난다. 유저는 인지하지 못하지만 코드가 실행되면 DOM 요소나 객체, 배열 등 특정 값을 읽고 쓸 때에도 비용이 발생하는데, 특히 DOM과 객체 같이 Depth가 있는 트리 구조 데이터의 경우 Depth가 깊어질수록 해당 값까지 찾아가는데 많은 비용이 발생한다. 때문에 자주 사용되는 값에 캐시를 사용하면, 탐색 비용이 줄어들어 그 만큼 성능적 이득을 볼 수 있다.

표 6-27 조건문 작성 예

잘못된 예

```
var padding = document.getElementById('result').style.padding;
var margin = document.getElementById('result').style.margin;
var position = document.getElementById('result').style.position;
```

올바른 예

```
var style = document.getElementById('result').style;
var padding = style.padding;
var margin = style.margin;
var position = style.position;
```

탐색비용을 절약할 수 있도록 캐시를 사용한다.

6.9 금지항목 및 주의사항

6.9.5 스타일 변경(1/2)

브라우저가 렌더링 되는 과정을 간단하게 설명하면 아래와 같다.

① HTML과 CSS를 파싱하여 DOM Tree와 Style 문맥을 생성한다.
② DOM Tree와 Style 문맥을 기반으로 엘리먼트의 색상, 면적 등의 정보를 가진 Render Tree를 생성한다.
③ Render Tree를 기반으로 각 노드가 화면의 정확한 비치에 표시되도록 배치한다.
④ 배치된 노드들의 visibility, outline, color 등의 정보를 시각적으로 표현한다.

③ 의 과정을 Reflow, ④의 과정을 Repaint라고 한다.
자바스크립트에서 DOM을 조작하면 브라우저 내부적으로 Reflow와 Repaint가 반복적으로 수행되는데, 이 두 과정이 서비스 성능 비용의 대부분을 차지한다. 결과적으로 이 두 과정에서 발생되는 비용을 얼마나 절감할 수 있는냐가 서비스의 성능을 결정한다.

표 6-28 조건문 작성 예 (계속)

```
<div id="container">
    <div id="sample" style="position:absolute;background:red; width:150px;height:50px">
        Sample BOX
    </div>
</div>
```

잘못된 예

```
function changeDivStyle() {
    var sampleEl = document.getElementById('sample');
    sampleEl.style.left = '200px';                  // reflow 1회, repaint 1회 발생
    sampleEl.style.width = '200px';                 // reflow 1회, repaint 1회 발생
    sampleEl.style.backgroundColor = 'blue';        // repaint 1회 발생
    }
// 총 reflow 2회, repaint 3회 발생
changeDivStyle();
```

6.9 금지항목 및 주의사항

6.9.5 스타일 변경(2/2)

올바른 예

```
// cloneNode를 사용하는 방법
function changeDivStyle() {
    var sampleEl = document.getElementById('sample');
    var clone = sampleEl.cloneNode(true);  clone.style.le
    ft = '200px';
    clone.style.width = '200px';  clone.sty
    le.backgroundColor = 'blue';
    document.getElementById('container').replaceChild(clone, sampleEl);    // reflow 1회, repaint 1회 발생
}
changeDivStyle();
```

```
// cssText를 사용하는 방법
function changeDivStyle() {
    var sampleEl = document.getElementById('sample');
    sampleEl.style.cssText = 'position:absolute;background:blue; width:200px;height:50p;left:200px;';    // reflow 1회, repaint 1회 발생
}
changeDivStyle();
```

참고

스타일 객체는 캐싱하여 사용하라

```
function changeDivStyle() {
    var sampleEl = document.getElementById('sample');
    var clone = sampleEl.cloneNode(true);
    var style = clone.style;
    style.left = '200px';  st
    yle.width = '200px';
    style.backgroundColor = 'blue';
    document.getElementById('container').replaceChild(clone, sampleEl);    // reflow 1회, repaint 1회 발생
}
changeDivStyle();
```

위의 코드는 이해를 돕기 위한 예시일 뿐, 실제 개발에서는 스크립트 코드 내에서 CSS를 직접 수정하지 말고, className을 수정하는 방법을 사용해야 한다.

6.9 금지항목 및 주의사항

6.9.6 이벤트 inline방식 금지(1/2)

자바스크립트 개발 초기에 많이 사용되던 방법으로 보통 doSomething()은 외부 자바스크립트 파일에 정의하는데, 만약 doSomething의 함수명을 변경하거나 버튼을 클릭했을 때 호출할 함수를 바꾸려면 자바스크립트와 HTML을 모두 변경해야 한다. 이렇게 inline 방식으로 사용할 경우, HTML로 자바스크립트가 서로 의존 관계를 만들어 작은 변경에도 수정 범위가 커지고 유지보수 및 디버깅이 어려워진다. 또한 inline 방식은 이벤트 핸들러를 1개밖에 사용할 수 없는 단점도 함께 가지고 있다.

표 6-29 조건문 작성 예

잘못된 예	`<button onclick="doSomething()" id="action-btn">Click Me</button>`
올바른 예	`<button id="action-btn">Click Me</button>`

inline 자바스크립트 코드는 HTML에 분리해서 사용한다.

6.9 금지항목 및 주의사항

6.9.6 이벤트 inline방식 금지(2/2)

```
// jQuery를 사용하는 경우
var btn;
function doSomething() {
    ...
}

btn = $('#action-btn').on('click', $.proxy(doSomething, this));
```

```
// jQuery를 사용하지 않는 경우
var btn;
function doSomething() {
    ...
}
function attachEvent(target, event, handler) {
    if (target.addEventListener) {
        target.addEventlistener(event, handler, false);
    } else if (target.attachEvent) {  target.attachE
        vent('on' + event, handler);
    } else {
        target['on' + event] = handler;
    }
}

btn = document.getElementById('action-btn'); attachEvent(btn, 'click', doSomething);
```

6.9 금지항목 및 주의사항

6.9.7 eval() 사용 금지

eval은 매개변수로 받은 문자열을 호출자의 지역 scope에서 즉시 실행하는 강력한 함수이다. 이 기능은 너무 강력해서 자바스크립트에서 가장 오용되고 있는 부분이기도 하다. eval에서 정의한 변수나 함수는 코드 파싱 단계에서는 문자열일 뿐, 실제 변수나 함수로 정의되는 단계는 실행 시점이다.

즉, 프로그램 실행 중 파서가 새로 기동되어야 하는데, 이는 상당한 부하를 만들어 프로그램 실행 속도를 현저히 느리게 한다.
뿐만 아니라, 사용자 입력 또는 네트워크로 들어온 문자열을 eval로 수행할 경우 서비스 전체에 심각한 보안 문제를 일으킬 수 있다.

참고

eval은 절대 사용하지 말아야 한다.

eval을 사용하는 거의 대부분은 사실 eval 없이도 만들 수 있다.

6.9 금지항목 및 주의사항

6.9.8 setTimeout, setInterval시 콜백 함수 문자열 전달금지

setTimeout, setInterval는 일정 시간 후에 첫 번째 파라메터로 받은 콜백 함수를 실행한다.
이때 콜백 함수인 첫 번째 파라메터는 문자열로도 전달 가능하지만, 문자열로 전달할 경우 내부적으로 eval로 처리되어 실행 속도가 느려진다.

표 6-30 조건문 작성 예

잘못된 예	올바른 예

function callback() {
  ...
}
setTimeout('callback()', 1000);
	function callback() {
	  ...
	}
	setTimeout(callback, 1000);
	
	setTimeout(function() {
	  ...
	}, 1000);

setTimeout, setInterval시 콜백 함수는 함수의 참조 또는 함수로 전달한다.

6.9 금지항목 및 주의사항

6.9.9 함수 생성자 new Function() 사용금지

많이 사용되는 방법은 아니지만 함수 생성자를 이용해서 함수를 선언할 수 있다.
하지만 이 경우, 문자열로 전달되는 파라메터가 수행시점에 eval로 처리되어 실행 속도가 느려진다.

표 6-31 조건문 작성 예

잘못된 예	올바른 예

`var doSomething = new Function('param1', 'param2', 'return param1 + param2;');`	`// 함수 선언식`
	`function doSomething(param1, param2) {`
	`  return param1 + param2;`
	`}`
	
	`// 함수 표현식`
	`var doSomething = function(param1, param2) {`
	`  return param1 + param2;`
	`};`

함수 선언 시 함수 선언식 또는 함수 표현식을 사용한다.

6.9 금지항목 및 주의사항

6.9.10 with() 사용금지

with문은 특정 객체를 반복적으로 접근할 때 간편함을 제공할 목적으로 만들어졌지만, 이런 의도와는 다르게 많은 문제점을 낳고 있어 사용하지 않는 것이 좋다.

표 6-32 조건문 작성 예

잘못된 예	올바른 예

with(document.getElementById('myDiv').style) {
background = "yellow";
color = "red";
border = "1px solid black";
}
	var style = document.getElementById('myDiv').style;
	style.background = 'yellow';
	style.color = 'red';
	style.border = '1px solid black';

특정 객체를 반복적으로 접근해야 한다면 새로운 변수에 캐싱하여 사용한다.

6.9 금지항목 및 주의사항

6.9.11 네이티브 객체 확장 및 오버라이드 금지(1/2)

자바스크립트는 동적인 특징이 있어, 언제든 무엇이든 수정할 수 있다. 다른 언어에서는 소스코드를 수정하지 않는 한 객체와 클래스를 수정할 수 없지만,
자바스크립트에서는 언제든 어떤 객체든 수정이 가능하므로 객체의 기본 동작이 예기치 않게 바뀔 수 있다. 이로 인해 네이티브 객체의 기본 동작을 기대한 개발자에게 혼란과 어려움을 줄 수 있으며, 이는 코드 내 예측할 수 없었던 오류를 만들 수 있다. 이런 동적인 특징은 언어 자체에서 강제할 수 있는 부분이 아니기 때문에, 개발자 스스로가 규칙을 세우고 따라야 하는 매우 중요한 부분이다. 설령 협업 개발자 간 합의된 변경이라 하더라도 브라우저 제조사가 이들 객체를 예고 없이 변경할 수 있고, 이런 변경이 기존 코드에 어떤 영향을 미칠지 아무도 예측할 수 없기 때문에, 합의된 변경 조차도 그 결과와 책임에서 절대 자유로울 수 없다.

표 6-33 조건문 작성 예

잘못된 예

```
Object.prototype.getKeys = function() {
    var keys = [];
    var key;
    for (key in this) {
        keys.push(key);
    }
    return keys;
};
```

올바른 예

```
function getKeys(obj) {
    var keys = [];
    var key;
    for (key in obj) {
        keys.push(key);
    }
    return keys;
}
```

네이티브 객체는 절대 수정하지 않는다.
만약 필요한 메서드가 있다면 네이티브 객체의 prototype에 작성하는 대신 함수로 만들거나 새로운 객체를 만들고 네이티브 객체와 상호작용하게 한다.

6.9 금지항목 및 주의사항

6.9.11 네이티브 객체 확장 및 오버라이드 금지(2/2)

참고

몽키패칭(monkey-patching)

네이티브 객체나 함수를 프로그램 실행 시 다른 객체나 함수로 확장하는 것을 몽키패칭이라 한다.

자바스크립트를 포함한 일부 라이브러리나 프레임워크에서 이런 식의 확장을 사용하고는 있다.

하지만 이는 캡슐화를 망치고 표준이 아닌 기능을 추가해 네이티브 객체를 오염시키므로 사용하지 말아야 한다.

이런 위험에도 불구하고, 신뢰성 있고 매우 중요한 몽키패칭의 특별한 한가지 사용법이 있는데, 바로 폴리필(polyfill)이다. 폴리필은 Array.prototype.map과 같이 자바스크립트 엔진에 새롭게 추가된 기능이 없는 경우, 비슷한 동작을 하는 다른 함수로 대체하는 것을 말한다.

폴리필과 같이 자바스크립트 기능의 호환성 유지 목적을 제외하고는 어떤 경우에도 네이티브 객체의 확장은 옳지 않다.

```
if (typeof Array.prototype.map !== 'function') {
    Array.prototype.map = function(f, thisArg) {
        var result = [];
        var i = 0;
        var n = this.length;
        for (; i < n; i += 1) {
            result[i] = f.call(thisArg, this[i], i);
        }
        return result;
    };
}
```

6.9 금지항목 및 주의사항

6.9.12 단항 연산자 사용금지

단항 연산자가 쓰인 연산의 결과를 한 눈에 파악하기 어렵다. 연산이 먼저인지, 값 할당이 먼저인지 고민해야 한다. 코드를 한 줄 더 줄이는 것보다 읽기 쉬운 코드를 작성한다

표 6-34 조건문 작성 예

잘못된 예	올바른 예

`var i = 0;`
`var num;`
 	
`for (; i < 10; i++) {`
 	
`}`
 	
`num = ( ++i ) * 10;`
	`var i = 0;`
	`var num;`
	
	`for (; i < 10; i += 1) {`
	`  ...`
	`}`
	
	`i += 1;`
	`num = i * 10;`

6.9 금지항목 및 주의사항

6.9.13 this에 대한 레퍼런스 저장금지

this는 함수 실행 시점에 결정된다. 어떤 함수 내부에서 또 다른 함수를 호출하면, 그 함수의 this는 상위 함수의 this와 같지 않다. 프로그래밍을 하다보면, 상위함수의 this에 대한 레퍼런스를 전달해야할 때가 있다. 비슷한 이름의 레퍼런스 변수(that, self, me 등)를 만들고, 내부 함수의 클로저로 사용하면, 상위함수의 this를 내부 함수에 전달할 수 있다. 이는 개발자에게 혼란을 줄 수 있는 방법이다. this를 결정하는 명확한 방식이 있으므로 그 방법을 사용해야 한다.

표 6-35 조건문 작성 예

잘못된 예

```
function() {
    var self = this;  ret
    urn function () {
        console.log(self);
    };
}

function() {
    var that = this;  re
    turn function () {
        console.log(that);
    };
}

function () {
    var _this = this;  re
    turn function () {
        console.log(_this);
    };
}
```

올바른 예

```
function printContext() {
    return function() {
        console.log(this);
    }.bind(this);
}

function printContext() {
    return () => console.log(this);
}
```

Function.prototype.bind 함수나 화살표 함수를 사용한다.

6.10 ES6+

6.10.1 변수 선언 시 "const", "let" 키워드 사용

"var"는 함수 스코프를 가진다. 코드 실행 전에 해당 변수가 hoist되므로 선언부보다 위에서 변수를 호출하여도 에러가 발생하지 않는다. 블록 스코프에서 "var"를 사용해도 블록 스코프 밖의 함수 스코프에 변수가 hoist되어 블록 스코프 밖에서 접근이 가능하다. 이는 Undefined된 변수의 프로퍼티에 접근할 때, 에러가 발생하는 잠재적인 원인이 된다. 반면에, "const"와 "let"은 개발자가 예상한대로 동작한다. 이 키워드를 통해 선언한 변수는 블록 스코프를 가진다. TDZ의 혜택을 받아 같은 스코프 내에서 선언 전에 호출하면 에러가 발생한다.

표 6-36 조건문 작성 예

잘못된 예	올바른 예

`var foo = 'foo';`
`var bar = 'bar';`
`...`
`bar = 'var';`
	`const foo = 'foo';`
	`let bar = 'bar';`
	`...`
	`bar = 'var';`

"const"를 사용해 변수를 선언하되, 할당 후 값이 변하면 "let"을 사용한다.

6.10 ES6+

6.10.2 "require", "module.exports“ 지향

자바스크립트에서의 모듈 패턴은 전역에서 특정 변수 영역을 보호하기 위해 사용된다. AMD, CommonJS 등의 모듈러가 보편화됨에 따라, ES6에서는 모듈 패턴을 지원하는 키워드인 import, export를 도입했다. 다만, 이 기능은 지원하는 브라우저의 점유율이 매우 낮기에 ES6 문법을 사용하기 위해서는 트랜스파일러를 사용해야 한다. import, export를 사용할 수 있는 환경이라면, require, module.exports 보다는 import, export를 사용하라. 사용할 수 없다면, "require", "module.exports"를 사용하라.

표 6-37 조건문 작성 예

잘못된 예

```
/* Top of file */
const StyleGuide = require('./StyleGuide');
 module.exports = StyleGuide.es6;
```

올바른 예

```
/* Top of file */
import StyleGuide from './StyleGuide';
export default StyleGuide.es6;

/* Top of file */
import { es6 } from './StyleGuide';
export default es6;
```

6.10.3 Import문 선언 위치

import문은 hoist 비용을 고려하여 코드의 맨 위로 작성한다.

표 6-38 조건문 작성 예

잘못된 예

```
import foo from 'foo';
foo.init();

import bar from 'bar';
```

올바른 예

```
/* Top of file */  imp
ort foo from 'foo';  im
port bar from 'bar';

foo.init();
```

6.10.4 제너레이터 사용 위치 주의점

제너레이터의 *의 위치가 바르지 않으면 트랜스파일링 되지 않는다. 따라서 가급적 제너레이터를 사용하지 않아야 한다. 꼭 사용해야 한다면 *의 위치에 주의한다. 반드시 function 바로 뒤에 써야 한다.

표 6-39 조건문 작성 예

잘못된 예

```
function * foo() {
    ...
}

const bar = function * () {
    ...
};

const baz = function *() {
    ...
};

function *foo() {
    ...
}

// very bad
function
*  foo
() {
    ...
}

// very bad
 const wat = function
*
() {
    ...
};
```

올바른 예

```
function* foo() {
    ...
}

const foo = function* () {
    ...
};
```

6.11 javascript 선언 타입

A. includ

<script>요소는 <head>요소 안에 쓰는 것이 일반적이다. 이런 형식을 취한 목적은 CSS나 자바스크립트와 같은 외부 파일 참조를 한 곳에서 처리하기 위함이다. 하지만 이런 형식은 CSS와 자바스크립트를 전부 다운로드하고, 파싱하고, 컴파일 할 때까지 페이지의 렌더링을 멈추게 한다. 브라우저는 <body> 요소를 만나게 되면서 렌더링이 시작되기 때문이다. 특히 자바스크립트를 많이 사용하는 페이지일수록 이런 지연은 눈에 띌 정도로 커진다.

표 6-40 javqscript include 예

잘못된 예

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Page</title>
<script src="../js/jquery-1.8.3.min.js"></script>
<script src="../js/common.js"></script>
<script src="../js/applicationMain.js"></script>
</head>
...
```

올바른 예

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Page</title>
<!-- 공통요소 head안, 맨 마지막에 씀 -->
<script src="../js/jquery-1.8.3.min.js"></script>
<body>
    ...
<!– 해당 페이지 요소 body안, 맨 마지막에 씀 →

 <script src="../js/common.js"></script></head>
 <script src="../js/applicationMain.js"></script>
 <script type="text/javascript">
 ...
</script>
</body>
</html>
```

6.11 javascript 선언 타입

B. jQuery 라이브러리 사용법

jQuery 같은 오픈 소스들은 어디서나 직접 접근이 가능한 url을 제공하는데, 외부 사정에 의해 url이 변경되거나 장애가 발생할 경우, 그 영향이 해당 서비스에 그대로 반영되어 장애로까지 이어질 수 있기 때문에 외부 url 대신 프로젝트 저장소에 소스 코드를 직접 다운로드해서 사용한다.

표 6-41 jQuery 라이브러리 사용 예

잘못된 예	올바른 예
<!DOCTYPE html> <html> <head> <title>HTML Page</title> <script src="http://code.jquery.com/jquery-1.8.3.min.js"%3E%3C/script%3E ...	<!DOCTYPE html> <html> <head> <title>HTML Page</title> <script src="../js/jquery-1.8.3.min.js"></script> ...

부록 A. 코딩 시 참고 사항

부록 A에서는 웹표준 기반의 마크업, 크로스 브라우징 범위를 설명한다.

A.1 웹표준 기반의 마크업

1. Table 기반의 마크업

웹표준 기반의 마크업이 도입되기 이전에는 구조와 표현을 모두 HTML 파일에 표현했다. 이를 위해 아래와 같은 방법을 사용했다.

- 구조와 표현을 모두 넣기 위해 <table> 엘리먼트를 사용
- 표현을 위한 <table> 엘리먼트를 추가하거나 중첩 사용
- 정밀한 표현을 위해 의미 없는 투명 이미지를 사용

이런 마크업 방법을 사용하면 HTML 파일의 용량이 커져서 유지보수 비용이 증가하고 효율성도 떨어진다. 사용자는 의미 없는 콘텐츠를 전달받아야 하며, 다양한 환경에서 접근할 수도 없다. 이러한 문제점들을 해결하기 위해 제안된 마크업 방법이 웹표준 기반의 마크업이다.

2. 웹표준 기반의 마크업

웹표준 기반 마크업이 Table 기반의 마크업과 차별화되는 점은 다음과 같다.

W3C 표준에 근거한 마크업

표준에 근거한 HTML과 CSS 마크업은 향후 웹 브라우저 호환성을 보장받을 수 있다.

의미에 맞는 HTML 엘리먼트를 사용하여 문서 구조 마크업

웹 문서의 내용을 HTML 엘리먼트의 의미만으로 구조화, 선형화하여 정보를 전달할 수 있다. 따라서 다양한 웹 브라우저와 장치에서있을 수 있으며, 화면 크기 등에 따라 디자인 정보를 가진 CSS 파일만 수정하면 One-Source Multi Use가 가능하다. 또한 웹 접근성이 높아져 언제, 어디서, 누구나, 어떤 디바이스, 어떤 응용 프로그램을 이용하더라도 동일한 정보를 제공받을 수 있다.

구조와 표현을 분리하여 마크업

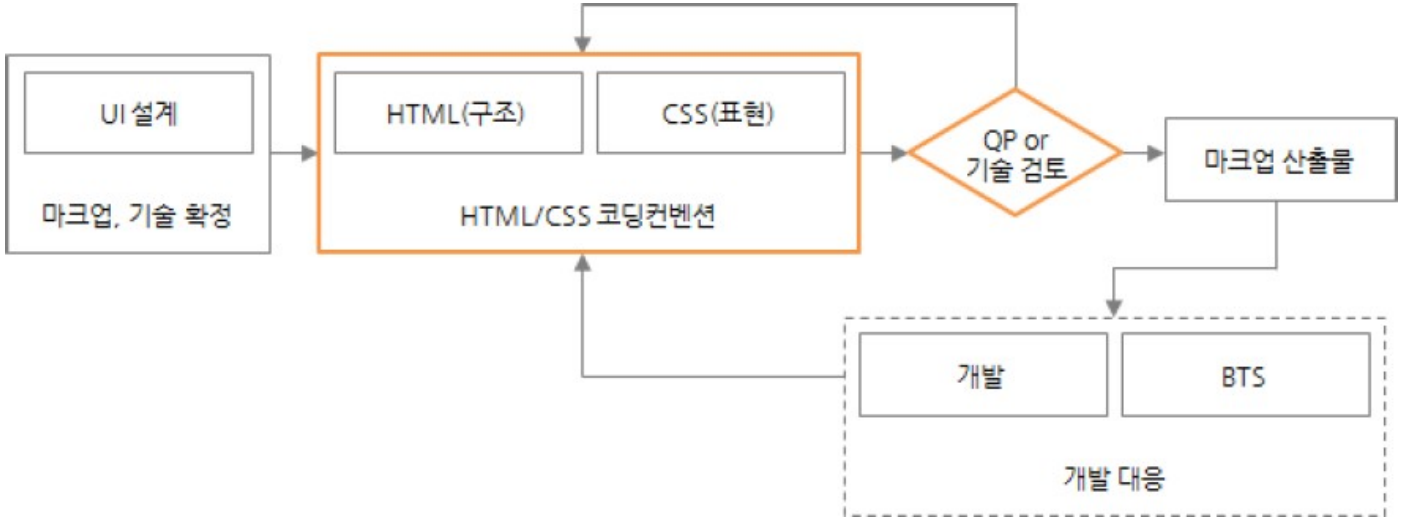
기존에는 구조와 표현이 분리되어 있지 않아서 디자인 정보만 수정하고 싶을 때도 전체를 수정해야 했다. 하지만 웹표준 기반의 마크업에서는 HTML은 문서의 메타데이터 정보를, CSS는 문서의 디자인 정보를 포함하도록 분리되어 있다. 디자인 정보를 수정할 때는 CSS 파일만 수정하면 되므로 유지보수가 한결 쉬워졌다. 또한, HTML 문서의 table 중첩 사용이 없어져 용량이 현저히 줄어들기 때문에, 로딩 시간을 단축할 수 있고 HTML 소스 코드의 재사용성이 높아진다.

A.1 웹표준 기반의 마크업

A.1.3 웹표준 기반의 마크업 프로세스

웹표준 기반의 마크업은 다음 그림과 같은 프로세스로 진행한다.

그림 A-1 웹표준 기반의 마크업 프로세스



모든 서비스 페이지 마크업은 웹표준 기반의 마크업 방법과 프로세스를 사용해야 한다.

A.2 크로스 브라우징 범위

크로스 브라우징 지원은 다음과 같이 3단계로 분류된다.

- 마크업 개발 범위: 마크업 개발 단계에 반드시 적용해야 하는 범위이다.
- QA 결함 대응 범위: QA에서 결함이 발생할 때 대응하는 범위로 마크업 스펙 질의/확정 단계에서 가감이 가능하다. 마크업 개발 범위를 포함한다.
- 서비스 예외 범위: 서비스 특성상 좀 더 확장된 사용자 층을 고려해야 하는 경우 적용해야 하는 범위이다. 특수 대상자가 아닌 일반사용자가 이용하는 페이지가 이에 해당되며, 마크업 개발 범위, QA 결함 대응 범위를 모두 포함한다.

다음 표는 크로스 브라우징의 범위를 나타낸 것이다. 인터넷 익스플로러를 제외한 나머지 브라우저는 최신 버전을 기준으로 한다.

표 A-1 크로스 브라우징 범위

운영체제	브라우저
윈도우 10	인터넷 익스플로러 10.x ~ Edge
	파이어폭스
	사파리
	크롬
	오페라
맥OS	파이어폭스
	사파리

A.3 단위 변환

표 A-2 단위변환

px	em	px	em
1px	0.07em	21px	1.5em
2px	0.14em	22px	1.57em
3px	0.21em	23px	1.64em
4px	0.29em	24px	1.71em
5px	0.36em	25px	1.79em
6px	0.43em	26px	1.86em
7px	0.5em	27px	1.93em
8px	0.57em	28px	2em
9px	0.64em	29px	2.07em
10px	0.71em	30px	2.14em
11px	0.79em	31px	2.21em
12px	0.86em	32px	2.29em
13px	0.93em	33px	2.36em
14px	1em	34px	2.43em
15px	1.07em	35px	2.5em
16px	1.14em	36px	2.57em
17px	1.21em	37px	2.46em
18px	1.29em	38px	2.71em
19px	1.36em	39px	2.79em
20px	1.43em	40px	2.86em

※ 참고 웹 사이트 : http://pxtoem.com/

부록 B. 예약어 목록

부록 B에서는 네이밍 예약어와 대체 텍스트 예약어 목록을 설명한다.

B.1 네이밍 예약어

표 B-1 공통 네이밍 예약어

의미	예약어	의미	예약어	의미	예약어	의미	예약어
검색	_srch_	왼쪽	_left_	이미지	img_	gnb	gnb
경로	_path_	오른쪽	_right_	블릿	bu_	lnb	lnb
그루핑 1	area_	위	_up_	화살표	arr_	snb	snb
그루핑 2	section_	아래	_dn_	별	star_	aside	aside
그루핑 3	box_	맨위	_top_	사진	pht_	sta	sta
그루핑 4	group_	맨아래	_btm_	백그라운드	bg_	답변/회신	_reply_
네이게이션	nav_	읽기	_read_	아이콘	ico_	재생	_play_
열기	_open_	쓰기	_write_	점	dot_	정지/멈춤	_stop_
닫기	_close_	수정	_modify_	선	line_	더보기	_more_
가기	_go_	취소	_cancel_	원	circ_	한국어	_kr_
이전	_prev_	삭제	_delete_	파일	fire_	영어	_en_
다음	_next_	설명	_dsc_	업로드	_upld_	중국어	_cn_
처음	_first_	썸네일	_thmb_	다운로드	_dwld_	일본어	_jp_
마지막	_last_	목록	_lst_	왼쪽 플로트	fl		
버튼	btn_	타이틀	_tit_	오른쪽 플로트	fr		
확대	_zin_	텍스트	_txt_	텍스트입력필드	input_txt		
축소	_zout_	페이징	pageing	라디오버튼	input_radio		
켜짐	_on_	찾기	_find_	체크박스	input_check		
꺼짐	_off_	등록	_reg_	파일입력필드	input_file		
오버	_over_	마스크	_mask_	주의/유의/참고/안내	_notice_		

B.1 네이밍 예약어

표 B-2 객체 예약어

분류	예약어	영역/객체
공통	#후-b	최상위 전역 네비게이션 영역 (Global Navigation Bar)
	.sta	서비스 이름, 연관서비스, 검색영역 (Service Title Area)
	.lnb	현재 서비스 지역 내비게이션 영역 (Local Navigation Bar)
	#snb	측면 내비게이션 영역 (Side Navigation Bar)
	#aside	문서의 주요 부분을 표시하고 남은 콘텐츠 영역
	.path	현재 페이지 경로
	.nav	내비게이션 엘리먼트
그루핑	.area	Section보다 높은 단계의 영역으로 특정 값으로 section을 제어할 때 그루핑
	.stn	heading 태그(h1~h6)를 포함한 엘리먼트들의 그루핑 (section)
	.box	section보다 낮은 단계의 heading 태그를 포함한 엘리먼트들의 그루핑
	.group	box보다 낮은 단계의 heading 태그를 포함한 엘리먼트들의 그루핑

[조합 기호: "_"]

B.1 네이밍 예약어

표 B-3 이미지 예약어

예약어	분류
tit_	제목
txt_	문장
gnb_, lnb_, snb_	내비게이션
tab_	탭
btn_	버튼
bu_	불릿
ico_	아이콘
line_	선
bg_	배경,박스
img_	이미지
_off, _over, _on	상태 변화

[조합 기호: "_"]

THANK YOU

WEB mindslab.ai

TEL +82-31-625-4340