KWEB, Korea University

2021 Web Application Essentials

Frontend Section

Written By: Lee, Chanyoung

May 17, 2021

Table of Contents

Chapter 1

Introduction to Front-end

Contents

1.1 Introducing WEB

WEB

웹(WEB)이란, 인터넷에 연결된 컴퓨터들을 통해 사람들이 정보를 공유할 수 있는 전 세계적인 정보 공간이다. 웹을 통해 공유되는 정보들은 대개 웹 페이지(web page)의 형태로 공유되며, 웹 페이지는 특수한 양식을 갖춘 텍스트로 구성된다. 각 웹페이지는 일반적인 텍스트(plain text)뿐만 아니라 이미지, 동영상, 다른 웹페이지로 연결되는 하이퍼링크(hyperlink) 등의 웹 자원(web resource)으로 다양하게 구성되고, 하나의 주제, 하나의 영역을 공유하는 여러 웹 자원과 웹 페이지는 웹 사이트(website)를 구성한다. 웹 사이트는 웹 서버(web server)라는 디바이스에서 동작하는 웹 애플리케이션(web application)이라고 하는 프로그램에 의해 구현되어 작동하며, 이 웹 애플리케이션으로 인해 권한만 충분하다면 전 세계 어디서든 웹 사이트에 접속할 수 있다.

본 교재에서 다루는 웹의 영역은 웹 애플리케이션과 관련된 영역이며, 흔히 웹 개발자가 다루는 웹의 영역도 이와 상당수 일치한다. 앞으로 여러분은 웹 애플리케이션의 작동 원리와 구조를 이해하고, HTML과 CSS, Javascript를 이용하여 웹을 디자인하고 설계하는 학습을 할 것이다.

Front-end and Back-end

웹 서버, 혹은 웹 사이트가 작동하는 원리를 생각해보자. 예를 들어, 사용자가 블로그 형태의 웹 사이트에 접속했을 때 사용자에게 보여지는 영역에는 어떤 것이 있는가? 블로그의 블로거, 블로거에 대한 상세 정보, 게시물, 게시물에 대한 상세 정보, 댓글, 댓글에 대한 상세 정보 등이 있을 것이다. 뿐만 아니라 블로그를 예쁘게 꾸민 디자인 등도 사용자에게 보여지는 영역이다. 이렇듯 사용자에게 보여지는 영역을 **front-end**라고 부르며, 사용자(client) 쪽에서 동작하는 영역이라고 하여 **client-side**라고도 부른다.

반대로, 사용자에게 직접적으로 보여지지 않는 영역도 있다. 사용자의 요청에 맞게 적절한 웹 페이지를 구성하여 전달해주는 로직, 게시물이나 댓글, 블로거의 정보를 데이터베이스에 저장하고 데이터베이스로부터 읽어오는 기능, 회원에 따라 글을 작성할 권한을 부여할 것인지, 댓글을 수정할 권한을 부여할 것인지 결정하는 로직 등이 있을 것이다. 이렇게 사용자에게 직접적으로 보여지지 않는 영역을 **back-end**라고 부르고, 이러한 로직은 서버상에서 동작하기 때문에 **server-side**라고도 부른다.

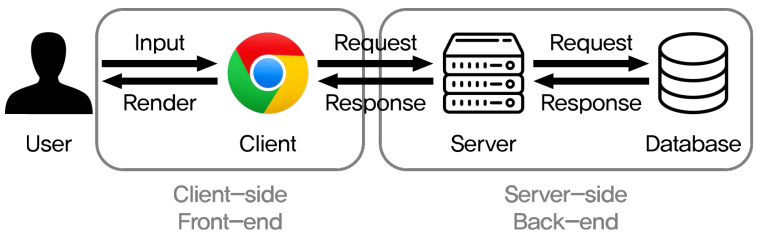


Figure 1.1 How Typical Web Application Works

본 교재에서는 front-end 분야를 다루며, front-end에서 기본적으로 사용되는 HTML, CSS, Javascript를 학습한다. 이를 응용하여 실무에서는 React.js, Vue.js, Angular.js 등의 프레임워크를 사용하기도 한다.

1.2 2021 Study Schedule

스터디 일정

Table 1.1 2021 Front-end Study Schedule

회차	스터디 날짜	스터디 내용	비고
1	03/29 ~04/02	Intro. to Front-end, HTML: The Basic Structure	
2	04/05 ~04/09	HTML: The Basic Structure, CSS: Designing HTML	중간고사 이전 회차
3	05/03 ~05/07	CSS: Designing HTML	중간고사 이후 회차
4	05/10 ~05/14	CSS: Designing HTML	
5	05/17 ~05/21	Basics of Javascript	
6	05/24 ~05/28	JS: Dynamic Frontend	
7	05/31 ~06/04	JS: Dynamic Frontend	기말고사 이전 회차

OUT COUNT 제도

준회원 스터디 과정에서는 3 OUT COUNT 제도가 시행되며, OUT COUNT는 다음과 같은 경우에 적용된다.

- 0.5 OUT : 스터디 사유 불참, 스터디 무단 지각, 과제 지각 제출
- 1.0 OUT : 스터디 무단 불참, 과제 미제출
- 2.0 OUT : 과제 Cheating

스터디 지각은 스터디 시작 이후 15분까지만 인정되며, 과제 지각 제출은 과제 제출 기한 이후 3일 이내에 제출되었을 경우에만 인정된다. OUT COUNT가 3.0을 초과하면 본 스터디의 Pass 자격이 박탈될 수 있다.

평가 시험

본 스터디는 평가 시험에 통과하여야 Pass할 수 있으며, 본 스터디를 Pass하여야 Back-end 스터디에 참여할 수 있다. 평가 시험 일정은 KWEB 종강총회 당일이며, 종강총회는 6/19(토)로 예정되어 있다. 평가 시험은 약 60분간 진행되며, 일정 점수 이상 취득하여야 통과할 수 있고, 통과하지 못하였거나 응시하지 않은 경우 여름방학에 보충 스터디[1]에 참여하여야 Back-end 스터디에 참여할 수 있다.

[1] 열리지 않을 수도 있음.

1.3 Setting Development Environment

Visual Studio Code

웹 개발을 보다 간편하게 할 수 있도록 개발 환경을 구축해보자. 준회원 스터디에서는 널리 이용되는 소스 코드 편집기인 Visual Studio Code를 사용한다. 물론, Atom이나 Vim 등 본인이 더 선호하는 편집기가 있다면 사용하여도 무관하나, 사용 과정에서 문제가 발생했을 때 도움을 제공하기는 힘들다는 점을 미리 말해둔다.

VS Code 홈페이지인 https://code.visualstudio.com/#alt-downloads에 접속하여 본인의 운영체제에 맞게 설치한다.

Chrome / Firefox

Chrome과 Firefox는 널리 사용되는 대표적인 인터넷 브라우저이다. Internet Explorer 역시 널리 알려진 인터넷 브라우저이나, 지속적으로 업데이트되는 웹 표준을 제때 따라오지 못해 개발 시에 다소 어려움을 유발하고, Chrome이나 Firefox가 더 편리한 디버깅 도구를 제공하므로 Chrome이나 Firefox를 사용하는 것을 권장한다.

Chapter 2

HTML: The Basic Structure

Contents

2.1 Introducing HTML

HTML은 HyperText Markup Language의 약자로, 웹 브라우저에 웹 페이지의 구조를 체계적으로 표현하는 컴퓨터 언어이다. HTML은 웹 페이지의 가장 기본적인 뼈대를 이루며, 어떤 front-end framework를 사용하던 HTML을 필수적으로 이해하고 있어야 한다.

HTML is not a Programming Language?

소프트웨어 개발과 관련된 밈(meme)을 접하다 보면 가장 자주 듣는 밈 중 하나가 바로 "HTML is not a programming language", 즉 HTML은 프로그래밍 언어가 아니라는 밈이다. HTML은 프로그래밍 언어가 아닌, 구조를 서술하기 위한 언어로, 쉽게 접할 수 있는 프로그래밍 언어인 C, Java, Python 등과 달리 조건문, 반복문, 변수 선언 등의 기능이 전혀 없다. 구조를 서술하기 위해 존재하는 언어이니만큼, **미리 정해진 규칙**에 따라 HTML을 작성하여 **구조를 표현**한다는 개념으로 접근하면 HTML이라는 언어를 쉽게 이해할 수 있을 것이다.

Figure 2.1 HTML is not a programming language

2.2 Basic Structure of HTML

Tags and Elements

HTML은 HyperText Markup Language, 즉 마크업 언어이며, 마크업 언어는 구조를 서술할 때 정해진 마크로 시작하여 마크로 끝나는 언어이다.

Code 2.1 Example of HTML

```
<!doctype html>
<html>
<head>
    <title>Example Domain</title>
    <meta charset="utf-8">
    <meta http-equiv="Content-type" content="text/html; charset=utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
</head>
<body>
    <div>
        <h1>Example Domain</h1>
    </div>
</body>
</html>
```

Code 2.1을 관찰하여 HTML의 구조를 이해해보자. Example Domain이라는 내용을 `<title>`과 `</title>`이 감싸고 있는데, 이러한 문자열을 태그(tag)라고 한다. 두 태그에서 `title`을 **태그 이름(tag name)**이라고 하며, 태그 내부의 부분(Example Domain)을 **내용(content)**이라고 한다. 유사하게, 이 `title` 태그를 `<head>`와 `</head>`가 감싸고 있다. 이렇게 태그와 그 태그 사이에 감싸져 있는 내용이 하나의 **HTML 요소(element)**를 구성하며, HTML 요소의 앞쪽에 위치하여 시작을 알리는 태그를 시작 태그, 뒤쪽에 위치하여 끝을 알리는 태그를 끝 태그라고 한다. 이처럼 HTML은 기본적으로 내용의 양 끝단을 태그로 감쌈으로써 구조를 표현한다.[1]

Code 2.1을 더 관찰해보면 `head` 태그의 요소와 `body` 태그의 요소가 `html` 태그의 요소 일부분인 것을 확인할 수 있다. 즉, HTML 요소는 내부에 또 다른 HTML 요소를 포함할 수 있고, 다른 HTML 요소의 일부가 될 수 있으며, 이와 같은 구조를 nested 구조라고 한다. HTML에서는 요소 A와 요소 B가 요소 C의 일부분을 구성할 때, 요소 A와 B를 요소 C의 **하위 요소(sub element)** 혹은 **자식 요소(child element)**라고 하고, 요소 C를 요소 A와 요소 B의 **상위 요소(super element)** 혹은 **부모 요소(parent element)**라고 한다. 또한, 요소 A를 요소 B의 **형제 요소(sibling element)**라고 한다.

또한, `meta` 태그에서는 태그 내에 태그 이름 외의 다른 문자열을 확인할 수 있다. 이는 HTML 요소의 속성과 관련된 문자열로, 첫 번째 `meta` 태그는 `charset` 속성의 값이 `utf-8`이며, 두 번째 `meta` 태그는 `http-equiv`의 값이 `Content-type`, `content`의 값이 `text/html; charset=utf-8`이라는 뜻이다.[2] 이렇듯 HTML의 속성은

[1] 다만, `meta` 태그와 같이 시작 태그만 존재하는 요소도 존재하며, 이러한 종류의 태그를 단일 태그라고 한다. 단일 태그는 그 자체만으로 하나의 요소가 된다.

[2] 이렇게 데이터의 정의나 이름과 데이터의 값으로 이루어진 데이터 집합을 **키/값 쌍(key-value pair)**라고 하며, 데이터의 정의를 키(key), 데이터의 값을 값(value)라고 한다. 예를 들어, 어떠한 학생의 인적 정보를 저장한 데이터 집합이 있고, 그 내부에는 학생의 이름이 홍길동, 학번이 2021990999, 성별이 남성, 거주지는 서울특별시 성북구 등의 정보가 포함되어 있다고 가정하자. 그러면 학생의 "이름", "학번", "성별", "거주지" 등은 key가 되고, "홍길동", "2021990999", "남성", "서울특별시 성북구" 등은 각 key에 대한 value가 된다. 웹 분야 뿐만 아니라 컴퓨터과학 분야 전반에서 사용되는 대부분의 데이터는 이러한 key-value pair 형태로 표현될 수 있다.

key-value pair 형태의 데이터 집합이며, **속성(attribute)**은 key, 속성값은 value에 대응된다. 속성값은 기본적으로 큰따옴표(") 내에 작성되며, 속성과 속성값은 등호(=)로 연결되며, 각 key-value pair는 공백을 사이에 두고 나열된다. HTML 요소의 속성은 태그나 내용 못지않게 중요한 역할을 하는데, 2.3절에서 더 자세히 다룬다.

이제까지 살펴본 HTML 요소의 구조를 종합해보면 **Figure 2.2**와 같이 표현될 수 있다.

Figure 2.2 Structure of HTML Element

HTML의 기본적인 구조

HTML 문서의 구조에는 가장 기본적인 틀이 정해져 있다. 먼저 `doctype`을 이용하여 문서가 HTML 문서임을 명시하고, 가장 상위 태그로 `html` 태그, 그 아래에 `head`와 `body` 태그가 위치해야 한다. 코드로 나타낸다면 **Code 2.2**와 같다.[3]

Code 2.2 Basic Structure of HTML

```
<!doctype html>
<html>
    <head>
        <!-- Head Element Content -->
    </head>

    <body>
        <!-- Body Element Content -->
    </body>
</html>
```

`head` 태그는 문서에 대한 전반적인 정보를 담고 있는 태그로, 이 태그에는 `title`, `meta`, `link` 등 다양한 태그들이 사용된다. 이번 절에서는 `title`과 `meta` 두 태그만 살펴보자. 먼저, `title` 태그 내부의 텍스트는 웹 페이지의 제목으로, 웹 브라우저를 통해 웹 페이지에 접속했을 때 브라우저의 상단에 표시된다. `meta` 태그는 HTML 문서의 인코딩 방법, viewport, 키워드 등 중요한 정보들을 지정할 수 있는 태그이다.

`body` 태그는 웹 페이지에서 사용자에게 보여질 부분이 포함되는 태그이다. 개발자가 웹 페이지를 통해 사용자에게 보여주고자 하는 텍스트, 이미지 등의 내용은 `body` 태그 안에 작성되어야 하며, 2.3절에서 본격적으로 웹 페이지의 내용을 작성하는 방법을 다룬다.

[3] HTML에서는 주석을 `<!--`, `-->`로 표현한다.

2.3 Commonly Used HTML Tags

이 절에서는 소개되는 태그들은 HTML에서 매우 자주 사용되는 태그들이다. 예제를 따라서 직접 작성해보고 웹 브라우저에서 열어 결과를 확인해보자.

Heading (`h1` - `h6`) Tags

`h1` - `h6` 태그는 heading의 약자로, 제목을 나타낼 때 사용되는 태그이다. h 뒤의 숫자가 작을수록 화면에 표시되는 글자의 크기가 크다.

Code 2.3 `h1` - `h6` Tags

```
<h1>This is h1 tag</h1>
<h2>This is h2 tag</h2>
<h3>This is h3 tag</h3>
<h4>This is h4 tag</h4>
<h5>This is h5 tag</h5>
<h6>This is h6 tag</h6>
```

`p`, `br` Tag

`p` 태그는 paragraph의 약자로, 문단을 구분해주는 태그이다. 이 태그는 문단과 문단 사이에 공백을 넣어주는 역할을 하며, 문단과 문단의 사이에는 별도의 공간이 생긴다.

HTML에서는 공백( ), 줄 바꿈(\n) 등의 whitespace 문자들은 종류와 관계없이 하나의 공백 취급을 받고, 이러한 whitespace 문자들은 여러 개 연속적으로 나열되어도 하나의 공백으로 취급된다. 따라서, 코드 내에서 줄 바꿈을 하여 작성해도 실제로는 줄 바꿈이 되지 않으며, 줄 바꿈을 하기 위해서는 `br` 태그를 줄 바꿈하고자 하는 위치에 넣어야 한다.

Code 2.4 `p`, `br` Tag

```
<p>
    HTML stands for HyperText Markup Language.<br>
    It is a language used to express the structure of webpage.
</p>
<p>
    Languages used to implement client-side are HTML, CSS and Javascript.<br>
    HTML and CSS are not programming languages, but Javascript is. <br>
</p>
```

`a` Tag

`a` 태그는 HTML 요소에 다른 웹 페이지로 이동할 수 있는 하이퍼링크를 걸 수 있는 태그이다. 이 태그는 속성값에 유의하여 작성하여야 하며, 다음 세 속성은 `a` 태그에서 가장 중요한 속성들이다.

- `href`: HTML 요소와 연결될 리소스의 주소
- `title`: HTML 요소와 연결될 리소스에 대한 설명. 호버(마우스를 위에 갖다 대는 것)를 했을 때 표시됨.
- `target`: 문서가 로드될 대상을 지정하는 옵션
  - `_blank`: 새로운 창이나 탭
  - `_self`: 현재 창
  - `_parent`, `_top`: 잘 쓰이지 않는 옵션

Code 2.5를 직접 실행해보고, 위의 속성값을 참고하여 `a` 태그를 활용해보자.

Code 2.5 `a` Tag

```
<a href="https://www.google.com" title="Google Homepage" target="_self">Google</a>
<br>
<a href="https://www.naver.com" title="Naver" target="_blank">Naver</a>
```

`button` Tag

`button` 태그는 클릭할 수 있는 버튼을 생성하는 태그이다. 이 태그를 눌렀다가 놓았을 때 특정한 동작이 수행될 수 있으며, `input` 태그를 이용해서도 유사한 기능을 구현할 수는 있으나 디자인적 관점에서 `button` 태그가 더 편하므로 더 많이 사용한다.

`button` 태그에 중요한 속성으로는 `disabled`, `type` 등이 있다. `disabled` 속성을 설정하면 버튼은 비활성화되어 클릭할 수 없는 상태가 되며, `type`은 버튼을 눌렀을 때 수행되는 동작의 성격에 차이가 생긴다. 버튼을 눌렀을 때 동작이 수행되도록 하는 방법은 5장에서 다룬다.

Code 2.6 `button` Tag

```
<button type="submit">Learn front-end</button>
<button disabled>Learn back-end</button>
```

List Tags

리스트, 즉 항목들을 나열하여 표현할 수 있는 태그들이 세 가지 있다. 순서가 없는 리스트(unordered list)를 나타내는 `ul` 태그, 순서가 있는 리스트(ordered list)를 나타내는 `ol` 태그가 있으며, 두 종류의 리스트 모두 리스트의 각 원소를 `li` 태그를 이용하여 나열한다.

`ul` 태그나 `ol` 태그를 이용하여 리스트를 표현하면 각 원소의 앞에 bullet과 같은 문자나 숫자가 자동으로 생긴다. 이러한 표식은 CSS를 활용하여 제거하거나 변경할 수 있다. **Code 2.7**을 직접 실행해보고, 다양한 형태의 리스트를 직접 만들어보자.

Code 2.7 List Tags

```
<h3>Unordered List</h3>
<ul>
    <li>Visual Studio Code</li>
    <li>Atom</li>
```

```
    <li>Sublime Text</li>
    <li>Vim</li>
</ul>

<h3>Ordered List</h3>
<ol>
    <li>Visual Studio Code</li>
    <li>Atom</li>
    <li>Sublime Text</li>
    <li>Vim</li>
</ol>
```

`img` Tag

`img` 태그는 HTML 문서에 이미지를 삽입할 수 있는 단일 태그이다. 다음은 `img` 태그에 사용되는 몇 가지 속성이다.

- `src`: 삽입하고자 하는 이미지의 주소
- `alt`: 이미지를 가져오는 데 실패하였을 때 대신 표시되는 이미지에 대한 설명
- `width`: 이미지를 띄우고자 하는 너비 (px)
- `height`: 이미지를 띄우고자 하는 높이 (px)

Code 2.8 `img` Tag[a]

```
<img src="https://s.pstatic.net/static/www/img/uit/2019/sp_search.svg"
     alt="Naver Logo"
     width=640>
<img src="https://t1.daumcdn.net/daumtop_chanel/op/nonexist.png" alt="Daum Logo">
```

[a] 이 코드에서는 HTML의 속성-속성값 쌍을 분리하기 위해 1칸의 공백 대신 개행 문자와 5칸의 공백을 사용하였다. HTML에서는 연속된 whitespace 문자들을 하나의 공백으로 취급하기 때문에 이렇게 표시할 수 있으며, 속성이 많거나, 속성값이 길거나, 내용이 길거나 등의 이유로 HTML 요소가 지나치게 길어지면 이와 같이 개행하여 작성하는 것이 좋다.

Font Tags

글자의 폰트를 변경할 수 있는 태그들이다. 간단한 HTML 문서를 작성할 때는 쓰일 수 있으나, CSS를 활용하여 작성하는 것이 권장되기 때문에 자주 쓰이지는 않는다.

Code 2.9 Font Tags

```
<b>Bold Text</b><br>
<strong>Important Text</strong><br>
<i>Italic Text</i><br>
<em>Emphasized Text</em><br>
<mark>Marked Text</mark><br>
<small>Small Text</small><br>
<del>Deleted Text</del><br>
<ins>Inserted Text</ins><br>
<sub>Subscript</sub> Text<br>
<sup>Superscript</sup> Text<br>
```

Input Tags

입력 태그들은 사용자로부터 정보를 입력받을 수 있는 태그들로, input, textarea, select 등이 있다. input 태그는 type 속성의 값에 따라 다양한 형태의 입력을 받을 수 있다. 다음은 input 태그의 type 속성의 값에 따라 입력받을 수 있는 정보의 형태 중 일부를 나타낸 것이다.

- text: 일반 텍스트 (plain text)
- password: 비밀번호
- radio: 선택 목록 중 하나만 선택할 수 있음
- checkbox: 선택 목록 중 여러 개 선택할 수 있음

textarea 태그는 input 태그와는 달리 여러 줄의 텍스트를 입력받을 수 있으며, 입력받는 부분의 크기나 설명 등은 속성을 이용하여 설정할 수 있다. select, option 태그는 여러 선택지 중 하나를 선택할 수 있는 drop-down 리스트를 만드는 태그이다.

Code 2.10 Input Tags

```
<h3>Input username: </h3>
<input type="text" name="username">

<h3>Input password: </h3>
<input type="password" name="password">

<h3>Gender: </h3>
<label><input type="radio" name="gender" value="male">Male</label>
<label><input type="radio" name="gender" value="female">Female</label>

<h3>Your Major: </h3>
<select name="major">
    <option value="cs">Computer Science</option>
    <option value="phy">Physics</option>
    <option value="chm">Chemistry</option>
    <option value="math">Mathematics</option>
</select>

<h3>Introduce yourself: </h3>
<textarea cols="40" rows="5"
          placeholder="Introduce yourself"
          name="introduction">
</textarea>
```

Code 2.10을 살펴보면, 모든 입력 태그에는 name 값이 지정되어 있고, 선택지가 있는 입력 태그에는 value 값이 지정되어 있다. name 속성의 값과 value 속성의 값은 key-value pair를 이루어, name의 값과 value의 값은 각각 입력받은 항목의 이름과 값을 뜻한다. 다만 텍스트를 입력받는 태그들의 경우 입력란에 입력한 값이 value 값이므로 value 값이 필요하지 않는데, 이 값을 직접 지정해줄 경우 기본값이 된다. name과 value의 값은 입력받은 데이터를 key-value pair의 형태로 전송할 때 사용되기 때문에 매우 중요한 속성이다.

HTML Entities

HTML에서는 특정 문자들이 구조를 표현할 목적으로 사용되기 때문에, 이러한 문자들을 실제로 웹 페이지 상에 표시하고자 할 때 문제가 발생할 수 있다. **Code 2.11**과 같이 작성된 HTML 문서는 어떤 문제가 발생할지 예상해보자.

Code 2.11 Improper Way of Displaying < and >

```
<h3>This is how inequality <signs> look like.</h3>
```

Code 2.11에서는 "This is how inequality <signs> look like."이라는 텍스트를 표시하려고 했던 의도와는 달리 `<signs>`라는 텍스트가 태그로 취급되어 "This is how inequality look like."이라는 텍스트가 표시된다. 이는 <, > 두 문자열이 태그를 나타내기 위한 문자이기 때문에 발생하는 현상이다. 그렇다면 이러한 문자들은 표시하기 위해서는 어떻게 작성해야 하는가? 또한, ※과 →와 같은 키보드 상에는 존재하지 않는 문자는 어떻게 작성하는가?

이러한 특수문자는 HTML 개체(entity)를 사용하여 나타낸다. HTML 개체는 개체의 이름을 사용하여 `&<entity-name>;`의 형태로 나타내거나, 개체의 번호를 사용하여 `&#<entity-number>;`의 형태로 나타낸다. 예를 들어, <, >의 개체 이름은 각각 lt, gt이므로 **Code 2.11**은 **Code 2.12**와 같이 나타내어질 수 있다.

Code 2.12 Proper Way of Displaying < and >

```
<h3>This is how inequality &lt;signs&gt; look like.</h3>
```

이외에도 `&quot;`("), `&amp;`(&) 등의 개체도 존재한다. 특히 ` `는 non-breaking space의 약자로, 공백을 뜻한다. 앞에서 `br` 태그의 필요성을 설명하면서 whitespace 문자는 여러 번 연속되어 나열되어도 하나의 공백으로 취급된다고 하였다. 그러므로 여러 칸을 띄워 표시할 때 단순히 space를 여러 번 입력하여 나타내면 안되고, ` `라는 HTML 개체를 이용하여 연속된 공백을 나타내야 한다. **Code 2.13**을 확인해보자.

Code 2.13 Usage of ` `

```
<h3>This is how you use     nbsp.</h3>
<h3>This is how you use     nbsp.</h3>
```

이 외에도 별도의 개체 이름이 없는 특수문자의 경우 유니코드 번호[4]를 이용하여 삽입할 수 있다. 예를 들어, reference mark, 혹은 흔히 당구장 문자(※)로 불리는 문자의 유니코드 번호는 8251이므로, **Code 2.14**와 같이 작성될 수 있다.

Code 2.14 Proper Way of Displaying < and >

```
<h3>&amp;#8251; This is how inequality &lt;signs&gt; look like.</h3>
```

[4]유니코드 번호는 https://www.fileformat.info/info/unicode/char/search.htm 페이지에서 검색하여 찾을 수 있다.

`div` and `span` Tag

HTML 코드의 가독성을 좋게 하고, CSS와 JS를 이용하여 웹 페이지를 디자인하고 추가 기능을 부여하기 위해서는 HTML을 구조화할 필요가 있다. 지금까지 살펴본 태그들은 모두 특정한 기능을 가진 태그들인데, 구조화할 때 사용되는 태그가 앞의 태그들처럼 특정한 기능을 갖는다면, HTML 코드의 구조가 엉망진창이 될 수 밖에 없다. 이러한 이유로 존재하는, 특정한 기능이 없고, HTML 요소들을 묶어서 레이아웃(layout)을 구성하기 위해 존재하는 태그를 non-semantic 태그라고 하며, `div`와 `span` 태그가 있다.

`div` 태그는 인접한 요소와 같은 줄에 있으려고 하지 않고, `span` 태그는 인접한 요소와 같은 줄에 있으려고 한다는 차이가 있다. **Code 2.15**를 통해 이러한 특징을 확인해보자.

Code 2.15 `div` and `span` Tags

```
<div>
    <h3>&lt;div&gt; tag</h3>
    <div>
        <textarea></textarea>
    </div>
    <div>
        <textarea></textarea>
    </div>
</div>

<div>
    <h3>&lt;span&gt; tag</h3>
    <span>
        <textarea></textarea>
    </span>
    <span>
        <textarea></textarea>
    </span>
</div>
```

Code 2.15를 조금만 관찰하면, 비슷한 기능이나 역할을 하는 요소들이 묶여있어 전체적인 구조를 파악하기 용이하다. 이렇게 `div`와 `span` 태그를 사용하는 것만으로도 그렇지 않았을 때보다 코드가 훨씬 구조화되고, 가독성이 대폭 좋아진 것을 확인할 수 있다.

`div`와 `span`은 대표적인 non-semantic 태그이지만, 개발자의 편의에 따라 예약되지 않은 태그 이름 사용하여 HTML 문서를 작성할 수 있다. 유사하게, 태그 속성 역시 예약되지 않은 속성의 이름과 속성값을 사용하여 HTML 문서를 작성하고, 이를 CSS와 JS에서 사용할 수 있다.

2.4 Class and Id Attributes

2.3절에서 HTML 문서를 작성하는 기본적인 방법에 대하여 학습하였다. 3장부터는 CSS와 JS를 배우면서 HTML에 적용하는 과정을 다루는데, CSS에서는 각 요소에 원하는 디자인을 적용할 수 있고, JS에서는 각 요소를 추가 및 삭제하거나, 그 속성을 수정하는 등의 작업을 할 수 있다. 이러한 CSS와 JS를 HTML 문서에 적용할 때, 특정 요소 혹은 특정 분류의 모든 요소에 CSS나 JS를 적용하게 된다.

2.3절에서 HTML 문서를 구조화하기 위해 `div`나 `span` 태그의 쓰임새에 대해 학습하였다. 그러나 태그만으로는 기능이나 역할 등 개발자가 원하는 분류 기준에 따라 HTML 요소들을 분류하는 것은 어려우며, HTML 문서가 매우 길어진다면 단순히 태그 이름만으로 요소들을 구분하는 것은 불가능하다.

이렇게 특정 기준에 따라 요소들을 분류하거나, 특정 HTML 요소를 지정할 때 필요한 속성이 **class**와 **id**이다. Class와 id는 모든 HTML 요소에 적용할 수 있으며, CSS, JS를 HTML과 연동할 때 매우 중요한 역할을 한다.

Class Attribute

먼저, class 속성은 HTML 요소들을 **특정한 기준에 따라 분류(classify)**할 때 사용되는 속성이며, class 속성의 값을 class name이라고 한다. 특정한 기준으로 분류하였을 때 하나의 묶음으로 묶이는 요소들에는 각각 같은 이름의 class를 사용한다. 하나의 HTML 요소는 여러 class를 가질 수 있고, 각 class name은 공백을 이용하여 구분한다. **Code 2.16**은 class 속성을 활용한 예제이다.

Code 2.16 Class Attributes Example

```
<div class="page-thumbnail new">
    <img src="/resources/week2_handout.jpg">
    <span class="page-title">
        <a href="/study/201R/3">Week 2 Handout</a>
    </span>
</div>
<div class="page-thumbnail">
    <img src="/resources/week1_asgmt.jpg">
    <span class="page-title">
        <a href="/study/201R/2">Week 1 Assignment</a>
    </span>
</div>
<div class="page-thumbnail">
    <img src="/resources/week1_handout.jpg">
    <span class="page-title">
        <a href="/study/201R/1">Week 1 Assignment</a>
    </span>
</div>
```

Id Attribute

Id 속성은 **특정한 HTML 요소 하나를 식별(identification)**하기 위해 사용되는 속성이다. 하나의 요소는 여러 id를 가질 수 없고, 특정 id의 값을 갖는 HTML 요소가 여러 개가 될 수 없다.[5] 다만, 각 HTML 요소는 class와 id를 동시에 가질 수 있다. **Code 2.17**은 id 속성을 활용한 예제이다.

Code 2.17 Id Attributes Example

```
<div id="article-form">
    <input id="article-title" name="title">
    <textarea id="article-content" name="content"></textarea>
    <button>Submit</button>
</div>
```

Naming Convention

Class name이나 id를 작성할 때 반드시 준수해야 하는 작명 규칙(naming convention)은 없다. 그러나 협업이나 유지보수 등 생산성의 향상을 위해 널리 통용되고 권장되는 규칙을 소개한다.[6]

- 대문자의 사용은 지양하고, 소문자로만 구성한다. 숫자로 시작하지 않는다.
- 이름은 class나 id의 의미에 잘 부합하여 어떠한 기준으로 지어진 이름인지 알기 쉽게 작명한다.
- 여러 단어의 조합은 하이픈(-)으로 연결하여 작명한다. (예: `multiple-words`)

[5] 이 규칙을 위반하더라도 HTML 문서는 정상적으로 렌더링된다.
[6] 아래 소개되는 작명 규칙보다 더 자세한 규칙은 다음 링크를 참조하길 바란다: https://bogmong.tistory.com/14

2.5 HTML Exercises

Problem 1: Generating Survey Page

KWEB 동아리 설문 조사에 쓰일 HTML 문서를 **Figure 2.3**과 같이 작성하여라. KWEB 로고와 페이스북 페이지의 주소는 아래와 같으며, HTML의 표준 구조를 준수하고, 각종 태그를 사용해서 구현한다. 제시되지 않았더라도 각 요소에 적당한 속성과 속성값을 적절히 부여하고, `div`, `span` 태그와 class, id 등을 적절히 활용하여 HTML 문서를 구조화하고, 가독성을 높인다.

- KWEB 로고: http://info.korea.ac.kr/_res/info/img/community/img_kweb.gif
- KWEB 페이스북 페이지: https://www.facebook.com/kwebfamily/

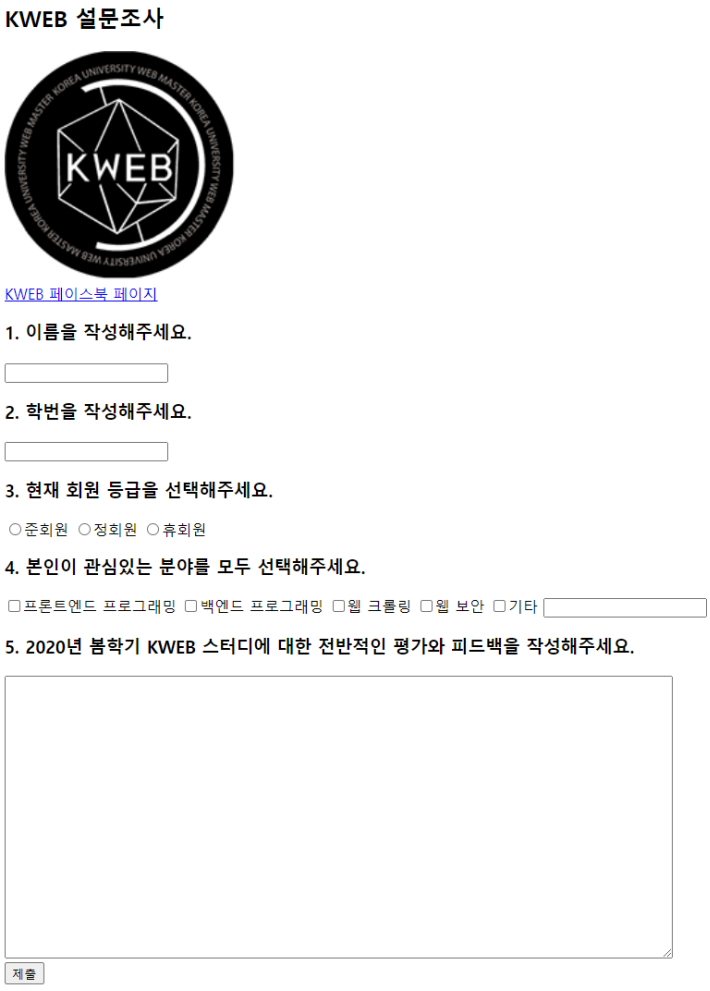


Figure 2.3 KWEB Survey Page Example

Problem 2: Structurizing HTML Code

2.2절에서 학습한 HTML의 기본 구조, 2.3절에서 학습한 `div`, `span` 태그와 2.4절에서 학습한 class, id를 이용하여, 앞의 **Code 2.10**을 HTML 표준에 맞게 수정하고, 자유롭게 구조화해보자. (정해진 정답은 없다)

Chapter 3

CSS: Desinging HTML

Contents

3.1 Introducing CSS

CSS는 Cascading Style Sheet의 약자로, HTML로 작성된 문서가 실제로 웹 브라우저에 어떻게 표현될지 명시해주는 컴퓨터 언어이며, CSS를 이용하여 HTML 문서의 지정된 요소에 의도하는 디자인을 적용할 수 있다. 주로 정적인 디자인을 명시하기 위해 사용되나, 각 요소의 상태나 웹 페이지가 표시되는 화면의 크기 등에 따라 동적인 디자인을 명시할 수도 있다.

CSS 등장 배경

과거 CSS가 존재하지 않았을 때는 HTML 문서에 웹 페이지의 구조뿐만 아니라 디자인 요소까지 작성하였다. 예를 들어, **Code 3.1**과 같이 `li` 태그는 정보를 저장하는 태그에 지나지 않았고, 텍스트에 스타일을 저장하기 위해 `font`, `b` 등의 태그를 사용하여 스타일을 저장하곤 했다. 그러나 시간이 흐르면서 스타일 및 레이아웃에 관한 정보를 훨씬 많이 저장하게 되었고, HTML 문서에는 본래의 목적인 "문서의 구조 서술"과는 거리가 먼, 디자인과 관련된 부가적인 정보가 지나치게 많이 작성되게 되었다. 이로 인해 HTML은 인간이 읽었을 때에도 문서의 구조를 파악하기 힘들고, 웹 브라우저가 사용자에게 웹 페이지를 렌더링하기 위해 분석하는 작업조차 힘든, 비효율적인 언어가 되었다.

HTML이 가지는 이러한 비효율적인 면을 개선하기 위해 1996년 CSS가 발표되었고, HTML과 CSS가 분리되면서 HTML에는 가급적 문서에 대한 구조만 서술하고, CSS에는 각 요소에 대한 스타일이나 레이아웃만을 서술하도록 권고되었다. CSS의 도입으로 HTML은 본연의 목적을 되찾아 문서의 구조를 표현하는 효율적인 언어가 되었으며, 더 나아가 웹 브라우저가 여러 웹 페이지에서 공통으로 사용되는 CSS 문서를 서버로부터 여러 번 다운로드할 필요가 없어져 웹 페이지 로딩 역시 빨라지게 되었다.

Code 3.1 Example of Early HTML

```
<body>
    <li><font color="red">HTML before CSS existence.</font></li>
    <b>This is a bold text. </b>
    <i>This text is italicized.</i>
</body>
```

3.2 Basic Structure of CSS

CSS의 기본적인 구조

CSS는 문서의 구조를 체계적으로 서술하는 언어인 HTML과 유사하게 스타일이나 레이아웃을 체계적으로 서술하는 컴퓨터 언어이므로 지켜서 작성해야 하는 규칙이 있다.

```
selector { property1: value1; property2: value2 }
```

Figure 3.1 Basic Structure of CSS

CSS의 구조는 **Figure 3.1**과 같이 표현될 수 있다. **선택자(selector)**는 HTML 요소를 태그 이름, 클래스 이름, 아이디, 상태, 속성 등을 기준으로 선택하는 방법을 서술하는 문자열이며, 선택자에 의해 선택된 요소들에 일괄적으로 적용할 스타일을 중괄호({}) 내부에 작성한다.

스타일은 key-value pair의 형태인 속성과 속성값의 집합으로 표현한다. **속성(property)**은 HTML 요소에 적용하고자 하는 디자인 요소로, 너비, 높이, 글자의 색, 폰트의 크기 및 굵기 등 250가지가 넘는 다양한 속성들이 존재하며, 각 속성에는 그에 대응하는 **속성값(value)**을 지정할 수 있다. 예를 들어, 글자의 색상과 관련된 속성은 `color`이며, 속성값으로는 `red`나 `blue`와 같은 색상 이름을 지정할 수 있다. 속성과 속성값 pair를 `property: value`의 형태로 쓰고, 각 pair를 세미콜론(;)으로 구분하여 나열한다.

Code 3.2 Simple Example of CSS

```
.main-panel {
    width: 800px;
    height: 450px;
    border: 1px solid black;
}

.ball {
    width: 80px;
    height: 80px;
    border: 1px solid red;
    border-radius: 40px;
    position: absolute;
}
```

하나의 요소는 여러 선택자에 의해 선택될 수 있으며, 이러한 선택자들에 의해 속성값의 충돌(conflict)이 일어날 수 있다. 이때 실제로 적용되는 속성값은 아래의 우선순위 규칙에 따라 우선순위가 가장 높은 선택자에 의해 적용되는 속성값이 적용된다.

1. 속성값의 뒤에 `!important`가 붙은 속성
2. HTML에서 `style` 속성을 사용하여 정의한 속성
3. Id > 클래스나 추상클래스의 이름 > 태그 이름 선택자의 속성
4. 상위 요소에 의해 상속된 속성

우선순위가 같은 경우, 부모-자식의 관계가 많을수록 우선순위가 높고, 그 다음으로는 나중에 작성된 속성이 적용된다.

Application of CSS on HTML

먼저 HTML 문서에 CSS 문서를 적용하는 세 가지 방법을 알아보자.

첫 번째 방법은 inline style으로, HTML 요소에 `style` 속성의 값으로 property-value pair를 직접 열거하는 방법이며, HTML 요소에 개별적으로 디자인을 적용하는 방식이기 때문에 선택자를 쓰지 않는다. 이 방법은 각 요소가 어떠한 디자인을 가지는지 쉽게 알 수는 있으나, 웹 페이지의 구조를 표현한다는 HTML의 목적에 위배되며, 요소마다 스타일을 작성해주어야 하므로 동일한 스타일을 적용하고자 하는 HTML 요소가 많아지면 문서가 불필요하게 길어지며 유지 및 보수 역시 번거로워진다. 따라서 inline style은 지양되는 스타일이지만, 예외적으로 서식이 있는 텍스트(rich text)를 표현할 때에는 자주 사용된다.

Code 3.3 Applying CSS with Inline Style

```
<div>
    Already member? <span style="font-weight: bold">Sign In</span>
</div>
<div>
    <span style="color: red">Sign Up</span>
</div>
```

두 번째 방법은 internal style sheet으로, `head` 태그 내부에 `<style type="text/css">` 요소를 삽입하고, 그 내부에 CSS 코드를 작성한다. 선택자를 사용할 수 있으므로 inline style보다는 효율적으로 작성할 수 있으나, HTML 본연의 목적에는 여전히 위배되며, 여러 HTML 문서에 동일한 CSS 문서를 적용할 때에는 여전히 번거롭고 비효율적이다.

Code 3.4 Applying CSS with Internal Style Sheet

```
<style type="text/css">
    .title {
        font-weight: bold;
        font-size: 16px;
    }
    #article-list {
        list-style-type: none;
        font-size: 12px;
    }
</style>
<div class="title">KWEB Front-end Study: </div>
<ul id="article-list">
    <li>Ch 1. Introduction to Front-end</li>
    <li>Ch 2. HTML: The Basic Structure</li>
    <li>Ch 3. CSS: Designing HTML</li>
    <li>Ch 4. Basics of Javascript</li>
    <li>Ch 5. Javascript: Dynamic Frontend</li>
</ul>
```

마지막 방법은 **external style sheet**으로, HTML 문서와 CSS 문서를 서로 다른 파일에 작성하고, HTML 문서의 `head` 태그 내부에 `link` 태그를 이용하여 CSS 파일을 연동한다. `link` 태그에는 다음과 같은 속성을 지정해주어야 한다.

- `type="text/css"` - 연결하고자 하는 문서가 CSS 형태임을 명시한다.
- `rel="stylesheet"` - 연결하고자 하는 문서가 HTML 문서의 stylesheet임을 명시한다.
- `href` - 연결하고자 하는 CSS 문서의 주소를 명시한다.

먼저 index.html과 style.css를 같은 폴더 내에 생성하고, **Code 3.5**와 같이 style.css를 작성한다.

Code 3.5 Applying CSS with External Style Sheet - style.css

```
.title {
    font-size: 16px;
    font-weight: bold
}

#article-list {
    padding: 0;
    list-style-type: none
}

#article-list > li {
    font-size: 12px
}
```

이후, index.html을 **Code 3.6**과 같이 작성한다. `link` 태그의 구조를 확인해보자.

Code 3.6 Applying CSS with External Style Sheet - index.html

```
<head>
    <link type="text/css" rel="stylesheet" href="./style.css">
</head>
<body>
    <div class="title">KWEB Study So Far: </div>
    <ul id="article-list">
        <li>Ch 0. Introduction to Front-end</li>
        <li>Ch 1. HTML: The Basic Structure</li>
        <li>Ch 2. CSS: Designing HTML</li>
    </ul>
</body>
```

이제 index.html 파일을 웹 브라우저에 열어서 확인해보면, CSS 파일에 작성된 디자인이 적용되었음을 확인할 수 있다. 이처럼 external style sheet 방식은 웹 페이지의 구조를 표현하고, 스타일을 표현한다는 HTML과 CSS 각각의 목적을 모두 달성하면서도 유지 및 보수가 용이하다는 장점이 있다.

3.3 Style Properties

3.2절에서 설명한 바와 같이 CSS의 속성은 매우 많다. 모든 속성과 속성값의 역할을 모두 알고 있는 것이 나쁘지는 않으나, 주로 사용되는 속성들을 잘 습득하고 그 외에 필요한 속성은 필요할 때 찾아서 사용하는 것이 바람직하다. 이번 절에서 스타일과 관련된 속성 중 자주 사용되는 속성들을 알아보자.

텍스트와 관련된 속성

- `text-align` – 텍스트를 수평적으로 어떻게 정렬할지에 관한 속성이다. 속성값으로는 `center`, `left`, `right`, `justify`가 있으며, 각각 가운데, 왼쪽, 오른쪽, 양쪽 정렬을 뜻한다.
- `text-decoration` – 텍스트를 선(line)을 이용하여 꾸민다. `text-decoration-line`, `text-decoration-color`, `text-decoration-style`의 3가지 세부 속성이 있고, 각 속성의 값을 `text-decoration` 속성의 값으로 차례대로 나열하여 표현하여도 된다.
- `line-height` – 줄 간격에 관한 속성. 100%는 `1`, 150%는 `1.5` 등 단위 없이 표현한다.
- `letter-spacing` – 글자 간 간격에 관한 속성. 단위는 %, em, px 등을 사용한다.
- `vertical-align` – 텍스트를 세로 방향으로 어떻게 정렬할지에 관한 속성. 속성값으로는 `baseline`, `top`, `bottom`, `text-top`, `text-bottom`, `middle` 등이 있다.

폰트와 관련된 속성

- `font-size` – 글자의 크기에 관한 속성. 단위는 em, px 등을 사용하여 표현한다.
- `font-weight` – 글자의 두께에 관한 속성입니다. `100`, `200`, ⋯, `900`의 값이 가능하다. 이 외에도 `normal`, `bolder` 등의 값이 가능하며, 각각 `400`, `700`에 해당하는 값이다.
- `font-family` – 글자의 서체를 지정하는 속성. 웹 브라우저에서 지원하는 서체뿐만 아니라 로컬 컴퓨터나 외부 서버에 있는 서체도 가능하다.
- `color` – 글자의 색상에 관한 속성. `red`, `blue` 등의 색상 이름이나, `rgb(26, 219, 158)` 또는 `rgba(26, 219, 158, .5)`와 같이 RGB(A) 형태로 나타낸 값, `hsl(161, 79%, 48%)`나 `hsla(161, 79%, 48%, .5)`와 같이 HSL(A) 형태로 나타낸 값, `#1ADB9E`와 같이 HEX 형태로 나타낸 값 모두 가능하다.

배경과 관련된 속성

- `background-color` – 배경의 색상에 관한 속성으로, 앞의 `color`과 동일한 형태의 값을 가질 수 있다.
- `background-image` – 배경에 이미지를 삽입할 수 있는 속성
- `background-repeat` – 배경 이미지가 반복되는 형태를 지정할 수 있다. 속성값으로는 `repeat`, `repeat-x`, `repeat-y`, `no-repeat` 등이 있다.
- `background-size` – 배경 이미지의 크기에 관한 속성

3.4 Selectors

3.2절에서 CSS의 구조에 관해 다루면서 선택자(selector)를 소개하였다. CSS에서는 선택자에 따라 원하는 HTML 요소에 원하는 스타일을 지정할 수 있다. 이번 절에서는 선택자를 작성하는 방법을 학습한다.

Universal Selector

전체 선택자(universal selector)는 HTML 문서의 모든 요소를 선택하며, `*`로 작성한다.

Code 3.7 Universal Selector

```
* { width: 80% }
```

Tag, Class, Id Selector

태그 이름, 클래스 이름, 아이디를 기준으로 요소를 선택하는 선택자로, 태그 이름은 `tag-name`, 클래스 이름은 `.class-name`, 아이디는 `#id`의 형태로 작성한다. 또한, 태그 이름, 클래스 이름, 아이디 등으로 표현된 선택자 `element`로 선택되는 요소 중 클래스 이름이 `class-name`인 요소를 선택하는 선택자는 `element.class-name`의 형태로 작성한다.

Code 3.8 Examples of Tag, Class, Id Selector

```
ul { list-style: none }
.title { font-size: 20px }
#article-list { padding: 0 }
.title.recent-article { font-weight: bold }
```

Child Selector and Descendants Selector

자식 선택자(child selector)와 자손 선택자(descendants selector)는 두 개 이상의 선택자를 이용하여 요소를 선택하는 선택자이다. 자식 선택자는 `parent > child`의 형태로 쓰며, `parent` 선택자로 선택된 각 요소의 바로 밑에 있는 자식 요소 중 `child` 선택자를 기준으로 선택한다. 반면, 자손 선택자는 `parent child`의 형태로 쓰며, `parent` 선택자로 선택된 요소들의 밑에 있는 모든 자식 요소 중 `child` 선택자를 기준으로 선택한다.

Code 3.9 Understanding Child Selector and Descendants Selector

```
<div class="class1"><span id="span1"></span></div>
<div class="class2"><span id="span2"></span></div>
<div class="class1">
    <div><span id="span3"></span></div>
</div>
```

Code 3.9를 통해 이해해보자. 자식 선택자인 `.class1 > span`은 `#span1`만 선택하나, 자손 선택자인 `.class1 span`은 `#span1`과 `#span3`를 선택한다.

Pseudo-class Selector

가상 클래스 선택자(pseudo-class selector)는 특정한 상태에 놓여있는 요소들을 선택하는 선택자이다. 예를 들어 `button` 태그로 구현한 버튼에 일반적인 상태, 호버된 상태(`:hover`), 비활성화된 상태(`:disabled`) 등의 상태를 클래스의 형태로 나타낸 것을 가상 클래스(pseudo-class)라고 하며, 각각 다른 스타일을 적용할 수 있다. 이러한 가상 클래스 선택자는 `:pseudo-class`의 형태로 쓰고, `element:pseudo-class`로 표현된 선택자는 `element` 선택자로 선택된 요소 중 `pseudo-class`에 해당하는 요소들을 선택한다. `a` 태그는 `:link`, `:visited`, `:hover`, `:active` 등의 가상 클래스를 가질 수 있다.

Code 3.10 Example of Pseudo-class Selector (1)

```
<style>
    button { color: black }
    button:hover { color: red }
    button:disabled { background-color: yellow }
</style>
<button>Enabled Button</button>
<button disabled>Disabled Button</button>
```

가상 클래스에는 HTML 요소의 상태뿐만 아니라 구조와 관련된 가상 클래스도 있다.

- `:first-child` – 선택된 요소 중 가장 첫 번째 요소를 선택하는 가상 클래스
- `:last-child` – 선택된 요소 중 가장 마지막 요소를 선택하는 가상 클래스
- `:nth-child(e(n))` – n으로 표현된 식 $e(n)$에 대해, 선택된 요소 중 $e(0), e(1), e(2), \cdots$ 번째 요소를 모두 선택하는 가상 클래스; `e(n)`은 $an + b$의 꼴만 가능하며, `e(n)` 대신 `odd`나 `even`을 사용할 수 있다.
- `:nth-last-child(e(n))` – `nth-child`와 유사하게 동작하나, 뒤에서부터 선택하는 가상 클래스

이 외에도, `:not(selector)`은 `selector`에 의해 선택되지 않는 요소들만 선택하는 가상 클래스이다.

Code 3.11 Example of Pseudo-class Selector (2)

```
<style>
    li:first-child { color: red }
    li:last-child { color: blue }
    li:nth-child(4n+3) { color: yellow }
    li:nth-last-child(4n+3) { color: green }
    li:not(#current) { font-style: italic }
</style>
<ul>
    <li>01</li>
    <li>02</li>
    <li>03</li>
    <li>04</li>
    <li>05</li>
    <li id="current">06</li>
    <li>07</li>
    <li>08</li>
    <li>09</li>
    <li>10</li>
</ul>
```

가상 클래스의 종류는 매우 많으므로, 자주 쓰이는 몇 가지를 제외하고는 필요할 때마다 찾아서 사용하면 된다.

더 많은 가상 클래스는 아래의 MDN reference에서 확인할 수 있다.

- https://developer.mozilla.org/en-US/docs/Web/CSS/Pseudo-classes

이번 절에서 다양한 선택자에 대하여 알아보았으며, 이 절에서 언급한 선택자 외에도 속성 선택자, 동위 선택자, 가상 요소 선택자 등 다양한 선택자가 존재한다. 교재에서 언급되지 않은 선택자들은 아래 제시된 링크들에서 확인할 수 있다.

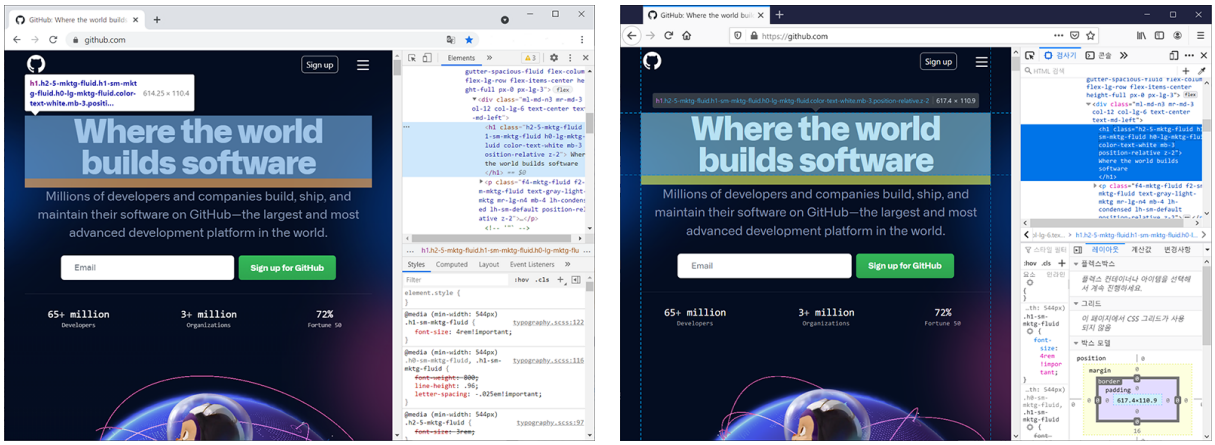
- https://www.w3schools.com/cssref/css_selectors.asp
- https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Selectors
- https://css-tricks.com/almanac/selectors

3.5 Layouts

3.1절에서 CSS는 스타일과 레이아웃을 표현하는 데에 목적이 있는 문서라고 소개하였다. 지금까지는 CSS를 이용하여 텍스트의 크기나 색상, 배경의 색상이나 이미지를 디자인하는 방법을 주로 다루었고, 이번 절에서는 CSS를 이용하여 레이아웃을 설정하는 방법을 다룰 것이다.

레이아웃이란

레이아웃(layout)은 직역하면 "배치"라는 뜻을 갖는 단어이다. CSS에서 레이아웃을 설정하는 것은 웹 페이지에서 사용자가 정보를 원활하게 주고받을 수 있게끔 HTML 요소들을 적절하게 배치하고 정돈하는 작업이다. HTML 요소들을 웹 페이지 상에 원하는 대로 배치하는 것은 문서가 복잡해질수록 어려운 일이다. 따라서 레이아웃을 정확히 설정하기 위해서는 각 CSS 속성들과 속성값의 성질을 정확히 알아야 한다.



(a) Chrome **(b)** Firefox

Figure 3.2 Checking layout of HTML element using DevTools

다행히도, 웹 브라우저의 개발자 도구는 특정한 HTML 요소가 웹 페이지 내에서 영역을 어떻게 차지하고 있으며, 어떻게 배치되었는지 확인할 수 있는 기능을 제공한다. Chrome과 Firefox에서 F12를 눌러 개발자 도구를 열고, Chrome은 Elements 탭을, Firefox는 검사기 탭을 눌러 열려있는 HTML 문서의 코드를 띄운다. 그리고 레이아웃을 확인하고자 하는 요소 위에 마우스를 hover하면 해당 요소의 레이아웃이 **Figure 3.2**와 같이 나타난다. 반대로, 웹 페이지에서 HTML 코드를 확인하고 싶은 부분을 우클릭한 후 [검사]를 누르면 해당 요소의 코드를 찾아준다. 이렇게 HTML 문서를 작성할 때 개발자 도구를 이용하면 특정 요소가 어떠한 레이아웃을 갖는지, 어떻게 수정해야 할지 등의 정보를 보다 쉽게 얻을 수 있다.

Box Model

HTML의 각 요소는 **사각형 형태**의 레이아웃을 가지며, 이러한 레이아웃을 box model이라고 한다. **Figure 3.3**을 참고하여 box model이 어떻게 구성되는지 알아보자.

- **Content**: HTML 요소의 내용에 해당하는 영역
  - 예: `img` 태그의 이미지, 제목 태그의 내부 텍스트

- **Border**: 요소의 테두리가 차지하는 영역
- **Padding**: Border과 content 사이의 영역
  - 예: 배경은 content 영역과 padding 영역에 적용됨
- **Margin**: 테두리 바깥쪽에서 요소가 차지하는 영역; 원칙상 다른 요소와는 겹치지 않음.

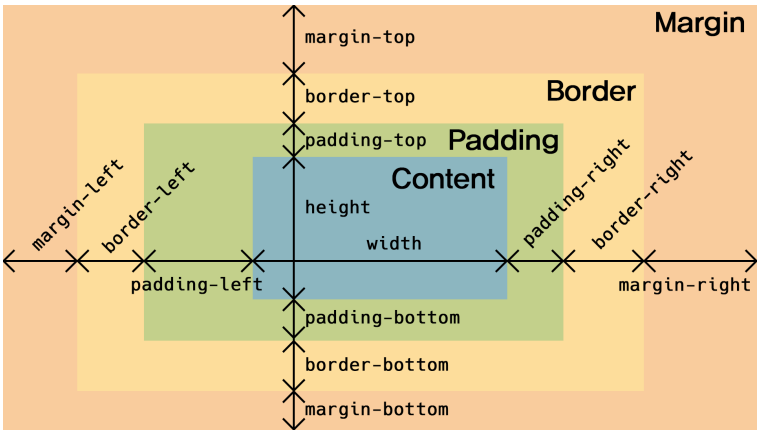


Figure 3.3 Box Model

이 box model은 HTML 요소 배치의 기본이 되기 때문에 제대로 이해하는 것이 매우 중요하다. 앞으로 나오는 예제 코드를 직접 작성해보고, 브라우저의 개발자 도구를 이용하여 padding과 border, margin이 어떻게 나타나는지 각각 살펴보자.

먼저, content에 관여하는 속성으로는 `width`와 `height`가 있다. 두 속성은 각각 content 영역의 너비와 높이를 지정하는 속성이며, px, em 등의 단위나 %를 이용하여 명시할 수 있다. %로 지정하는 경우는 상위 요소의 너비나 높이를 기준으로 한다.

Code 3.12 Content Area

```
<style>
    #small-box {
        width: 400px;
        height: 200px;
        background-color: red;
    }

    #large-box {
        width: 800px;
        height: 400px;
        background-color: blue;
    }
</style>

<div id="small-box"></div>
<div id="large-box"></div>
```

Border에 관여하는 속성으로는 `border-width`, `border-style`, `border-color` 등이 있다. `border-height`는 테두리의 굵기, `border-style`은 테두리의 모양, `border-color`는 테두리의 색상을 나타내는 속성으로, 테두리의 모양은 `solid`, `dashed`, `dotted` 등의 값을 지정하여 테두리의 모양을 결정할 수 있다. 이 세 가지 속성은 각각

따로 작성하여도 되고, `border` 속성에 굵기, 모양, 색상 순으로 나열하여 작성할 수도 있다. **Code 3.13**에서 `#box` 요소의 `border` 속성이 이러한 방법으로 작성되었다.

또한, 방향에 관한 키워드[1]를 사용하여 네 면 중 하나의 면에만 특정 스타일을 지정해줄 수 있다. 예를 들어, 왼쪽 면에만 스타일을 적용하고 싶은 경우 `border-left` 속성에 스타일을 작성한다.

`border-radius`는 테두리의 모서리를 둥글게 만들 수 있는 속성으로, 속성값이 하나인 경우 주어진 값을 반지름으로 하는 원형으로, 두 개인 경우 두 값을 짧은 반지름과 긴 반지름으로 하는 타원형으로 만든다. `border`와 마찬가지로 네 모서리에 모두 적용되며, 한쪽 모서리에만 적용하고 싶은 경우 역시 방향과 관련된 키워드를 사용하여 기술한다. 예를 들어, 좌상단 모서리의 스타일은 `border-top-left-radius` 속성에 작성한다.

Code 3.13 Border Area

```
<style>
    #box {
        width: 300px;
        height: 300px;
        border: 3px solid black;
        border-radius: 20px;
    }
</style>

<div id="box"></div>
```

Padding 속성은 border 영역의 각 면이 content 영역의 각 면으로부터 어느 정도 떨어져 있는지, margin 속성은 요소 가장자리의 각 면이 border 영역의 각 면으로부터 어느 정도 떨어져 있는지 명시하여 작성해준다. 앞의 border 영역과 마찬가지로 방향 키워드를 사용하여 한쪽 면에만 특정 padding/margin 값을 적용할 수 있다. (예: `padding-left`, `margin-bottom`)

다만 `padding`과 `margin` 속성은 방향마다 속성과 속성값을 모두 작성하는 번거로운 방식 대신, **Table 3.1**과 같이 `padding`, `margin` 속성의 값에 차례대로 각 방향의 값을 나열하여 shortened form으로 작성하는 것이 가능하다.

Table 3.1 Shortened `padding` and `margin`

Shortened Form	top	right	bottom	left
`padding/margin: A;`	A	A	A	A
`padding/margin: A B;`	A	B	A	B
`padding/margin: A B C;`	A	B	C	B
`padding/margin: A B C D;`	A	B	C	D

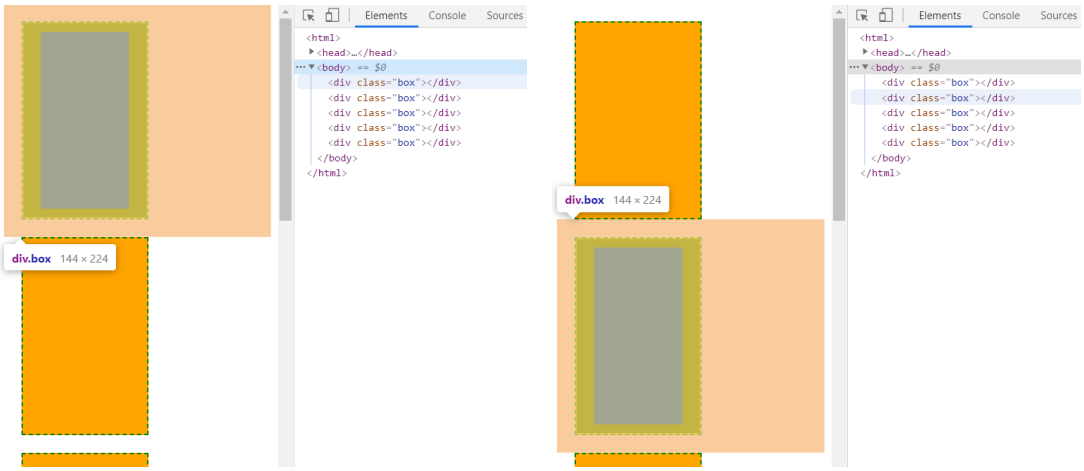
Code 3.14를 참고하여 `padding`과 `margin`을 정하는 방법을 이해하고, 개발자 도구를 이용하여 box model의 각 부분이 웹 브라우저상에 어떻게 표시되는지 확인해보자.

[1] CSS에서 문서의 위, 아래, 왼쪽, 오른쪽 방향의 키워드는 각각 `top`, `bottom`, `left`, `right`이다.

Code 3.14 Examining Box Model

```
<style>
    .box {
        width: 100px;
        height: 200px;
        border: 2px dashed green;
        padding: 10px 20px;
        margin: 20px;
        background-color: orange;
        display: inline-block;          /* We will learn this later */
    }
</style>

<div class="box"></div>
<div class="box"></div>
<div class="box"></div>
<div class="box"></div>
<div class="box"></div>
```

Code 3.14에서 `display: inline-block;`을 지우고 웹 페이지를 열어보면 각 `.box` 요소가 세로로 나열된다. 이때 **Figure 3.4**와 같이 두 요소의 margin이 겹친 것을 확인할 수 있으며, 이렇게 HTML 요소 간에 margin이 겹치는 현상을 마진 상쇄 또는 마진 겹침(margin collapsing)이라고 한다. Margin 겹침 현상은 인접한 형제 요소 간의 상하 margin이 겹칠 때, 빈 요소의 상하 margin이 겹칠 때, 부모 요소의 top(bottom) margin과 첫 번째(마지막) 자식 요소의 top(bottom) margin이 겹칠 때 발생한다. **Code 3.14**의 경우는 형제 요소 간의 상하 margin이 겹친 경우로, 이러한 현상에 유의하여 디자인하여야 한다.



(a) Upper Box **(b)** Lower Box

Figure 3.4 Margin Collapsing

Inline-level and Block-level Elements

지금까지 모든 HTML 요소는 box model에 따라 영역이 결정되고, 영역에 따라 다른 요소들과 맞물려 배치됨을 학습하였다. 앞의 **Code 3.14**에서는 HTML 요소가 이전 형제 요소의 오른쪽에 충분히 빈 공간이 있다면 오른쪽에, 그렇지 않으면 아래쪽에 배치되었다. 그런데 지금까지 HTML을 다루면서, 이전 형제 요소에 따로 margin이나 padding을 지정해주지 않아도 이전 형제 요소와 같은 줄에 배치되지 않는 현상을 보았다.

Code 3.15 Example of Inline-level, Block-level Elements

```
<h1>HTML Element Layout</h1>
<h2>This paragraph is about <span style="color: red">HTML element layout</span>.</h2>
```

Code 3.15에서 h1 태그와 h2 태그에 padding이나 margin이 전혀 부여되지 않았음에도 불구하고 서로 다른 줄에 배치되었다. 반대로, h2 태그 내부의 텍스트와 span 태그 내부의 텍스트는 서로 같은 줄에 배치되어 있다. 앞의 2.3절에서 non-semantic 태그는 다른 요소와 같은 줄에 있으려고 하는지에 따라 div 태그와 span 태그로 나뉜다고 했던 것을 기억하는가? 이처럼 같은 줄에 다른 요소가 배치되는 것을 허용하는지에 따라 inline-level 요소와 block-level 요소로 구분할 수 있고, 이 성질은 display 속성을 이용하여 명시할 수 있다.

먼저 **inline-level 요소**는 다른 요소와 같은 줄을 공유하고, content의 너비만큼만 가로 폭을 차지한다. Inline-level 요소에는 width, height, margin-top, margin-bottom, padding 등의 속성을 적용할 수 없으며, 그 내부에 inline-level 요소를 포함할 수 없다. display 속성값이 inline인 요소는 inline-level 요소의 성질을 가지며, span, a, strong, img, br, input, select, textarea, button 등의 태그는 기본적으로 inline-level 요소이다.

반대로 **block-level 요소**는 다른 요소와 같은 줄을 공유하지 않으며, 웹 페이지 화면의 가로 폭을 모두 차지하여 여러 block-level 요소들을 배치하면 수직 방향으로 배치된다. Block-level 요소에는 width, height, margin, padding 등의 속성을 적용할 수 있으며, 그 내부에 inline-level 요소를 포함할 수 있다. display 속성값을 block인 요소는 block-level 요소의 성질을 갖게 되며, div, h1~h6, p, ol, ul, li, hr, table, form 등의 태그는 기본적으로 block-level 요소이다.

그런데, 레이아웃을 구성하다 보면 요소들을 같은 줄에 배치하면서 width, height, margin, padding 등의 속성을 사용해야 하는 경우가 있다. 이러한 문제는 해당 요소의 display 값을 inline-level 요소의 성질과 block-level 요소의 성질을 모두 갖는 inline-block으로 지정하여 해결할 수 있다. 또한, display 값을 none으로 지정하면 해당 요소가 보이지 않아, 사용자로부터 숨길 수 있다.[2]

Code 3.14에서 display 속성의 값을 바꿔보며 이해해보자.

Position

지금까지는 이전의 형제 요소가 배치되고 난 다음 공간에 다음 요소가 차례대로 배치되는 레이아웃 배치 방식에 대해 알아보았다. 이러한 방식이 일반적인 배치 방식이지만, 간혹 웹페이지에서 다른 요소들과의 위치와는 관계없이 고정된 위치에 배치되는 요소들이 있고, 다른 요소와 겹치게끔 배치되는 요소들이 있다. 대표적으로는 뉴스 기사에서 스크롤을 내려도 계속 화면상에 표시되는 광고들이 이러한 배치를 갖는다. 이렇게 특정 위치에 HTML 요소를 배치하고자 할때 position 속성을 이용한다.

position 속성에는 static, relative, absolute, fixed의 네 가지 값이 존재한다. 기본값은 static으로, static 배치 방식은 HTML 요소가 지금까지 학습해왔던 배치 방식, 즉 빈 공간에 왼쪽에서 오른쪽으로, 위에서 아래로 차례대로 배치된다. 나머지 세 값은 네 개의 방향 속성을 함께 사용하여 요소의 위치를 정할 수 있다. 방향 속성, 즉 top, left, right, bottom 속성은 기준점이 요소로부터 어떤 방향으로 얼마나 떨어져 있는지

[2] 유사한 속성으로는 visibility: hidden이 있으나, display: none과는 달리 요소가 공간을 차지하되 보이지만 않게끔 한다.

나타내며, 속성값이 명시되지 않은 경우 기본값은 0이다.

먼저 relative는 static으로 지정되었을 때를 기준으로 방향 속성의 값에 따라 이동되어 배치되며, 네 방향 속성이 모두 지정되지 않은 경우 static과 동일하게 배치된다. 요소의 위치가 relative로 인해 static이었을 때와 비교하여 벗어났다고 해서 다른 요소들의 위치가 바뀌지는 않는다.

absolute는 원래 요소가 배치되었어야 할 공간과 관계없이 position 값이 static이 아닌 부모나 조상 요소를 기준으로 배치되며, 이러한 부모나 조상 요소를 찾지 못하면 body 태그를 기준으로 배치된다. absolute 속성값을 갖는 요소가 원래 배치되었어야 하는 공간에는 다른 요소가 들어올 수 있으며, 요소의 display 속성이 block이더라도 너비가 content에 맞게 바뀌기 때문에 적절한 너비를 지정해주어야 한다.

fixed는 absolute와 유사한 성질을 갖지만, 위치를 지정하는 기준이 viewport, 즉 화면에 보여지는 웹페이지의 영역이다. 따라서 화면을 스크롤하여 올리거나 내렸을 때도 화면에서 고정된 위치에 요소를 배치할 때 사용된다.

Code 3.16에서 #pos-element에 position과 관련된 속성과 그 값을 지정하고 바꿔가며 position에 대해 정확히 이해해보자.

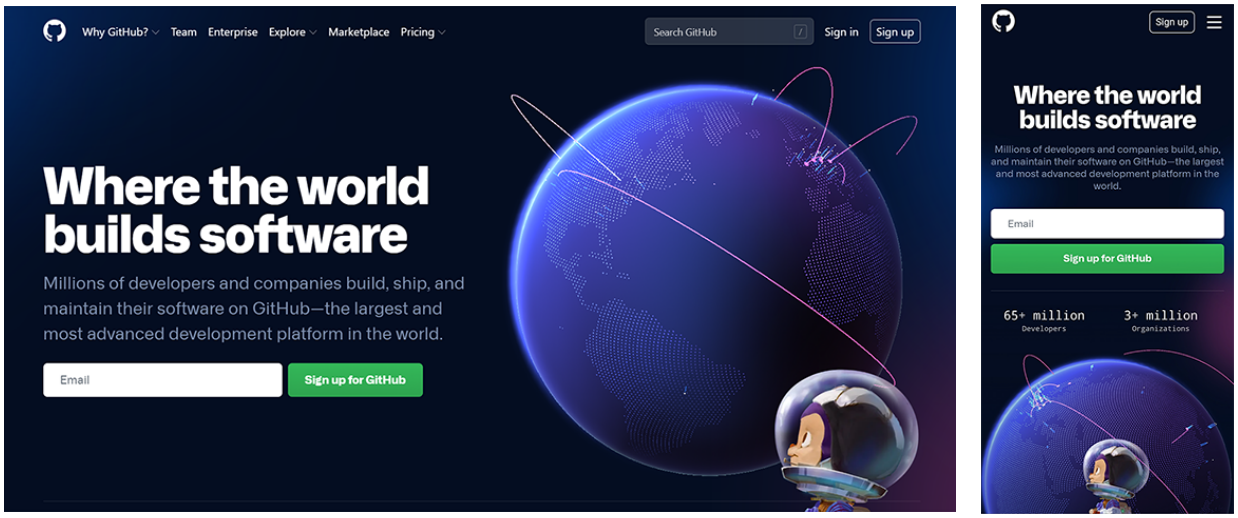
Code 3.16 Example of position Property

```
<style>
    .box {
        width: 100px;
        height: 3000px;
        border: 1px solid;
        display: inline-block;
        background-color: skyblue;
        border-color: blue;
    }

    .red-box {
        height: 100px;
        background-color: pink;
        border-color: red;
    }

    #pos-element {
        /* Your CSS code here */
    }
</style>

<div class="box">1</div>
<div class="box red-box" id="pos-element">2</div>
<div class="box">3</div>
<div class="box">4</div>
```

여담으로, position 속성을 다루다 보면 요소 간에 겹침이 발생하여 특정 요소를 다른 요소 위에 오도록 할 필요가 있다. 이때 z-index 속성을 이용하면 요소 간의 쌓임 순서를 조절할 수 있다. z-index의 값은 정수(integer)이며, 0이 기본값이고, 값이 높은 요소가 위에 쌓인다.

3.6 Responsive Web

지금까지 CSS를 이용하여 문서를 디자인할 때 주로 가로 폭이 넓은, PC의 웹 브라우저를 기준으로 작업해왔다. 그러나 모든 사용자가 항상 웹 페이지를 PC와 같이 가로 폭이 넓은 디바이스에서 열람하는 것은 아니다. 휴대폰과 같은 모바일 디바이스는 가로 폭이 좁아서 PC를 기준으로 설계한 웹 페이지는 모바일에서 열람했을 때 가독성이 심각하게 저하될 수 있다. HTML 요소들이 의도와는 다르게 배치될 수 있고, 이를 방지하고자 요소의 너비 등을 정해진 값으로 딱딱하게 정하면 모바일 디바이스에서는 좌우로 스크롤하면서 웹 페이지를 읽어야 한다. 웹 페이지는 가급적 좌우 방향으로는 움직이지 않고, 상하 방향으로만 움직여 정보를 전달하게끔 설계하였을 때 가독성이 좋은데, 위와 같이 PC를 기준으로 웹 페이지를 설계하면 가독성이 매우 떨어진다. 따라서, 웹 페이지가 렌더링되는 화면의 크기에 따라 디자인이 바뀌는, **반응형 웹 페이지(Responsive Web)**를 디자인할 필요가 있다.



(a) Desktop Browser **(b)** Mobile Browser

Figure 3.5 Webpage view of GitHub Homepage

Viewport

반응형 웹 페이지는 웹 페이지가 렌더링되는 화면, 즉 뷰포트에 따라 디자인이 바뀌므로 먼저 뷰포트를 설정해 주어야 한다. 뷰포트의 크기는 **Code 3.17**과 같이 HTML 문서의 `meta` 태그에서 설정한다.

Code 3.17 Viewport Setting

```
<meta name="viewport" content="width=device-width, initial-scale=1" >
```

Code 3.17의 코드는 페이지의 너비를 디바이스 화면의 너비로 설정(`width=device-width`)하고, 원래 페이지의 크기를 그대로 사용(`initial-scale=1`)하는 코드이다. 이 코드가 가장 기본적인 설정이며, 추가 설정이 가능하지만 이 교재에서는 기본 설정만 사용한다. 이러한 설정은 2.2절의 **Code 2.1**에서도 확인할 수 있다.

뷰포트는 개발자 도구를 이용하여 조절할 수도 있다. Chrome은 개발자 탭에서 Elements 탭 왼쪽에, Firefox는 오른쪽 상단에 모바일 디바이스와 유사한 아이콘(Toggle device toolbar)을 눌러 뷰포트를 조절할 수 있다.

Media Query

이제 CSS를 이용하여 웹 페이지를 반응형으로 디자인해보자. 반응형 웹페이지를 디자인하기 위해서 `@media query`라는 구문을 사용한다.

Code 3.18 Media Query Statement

```
@media only screen and (min-width: 800px) {
    /* CSS code goes here */
}
```

Code 3.18은 `@media query` 구문의 예시이다. 중괄호 내부에는 일반적인 CSS 코드를 작성하고, `@media query` 구문은 중괄호 내부의 디자인을 적용할 조건을 제시한다. 코드에서 `min-width: 800px`은 화면의 "너비가 800px 이상"이라는 조건이며, 이 외에도 다양한 조건을 제시할 수 있으나 **Code 3.18**이 가장 기본적인 형태이다.

`@media query` 구문을 이용하여 반응형 웹페이지를 디자인할 때 너비가 작은 화면에서 큰 화면의 순서로 작성하는 것이 원칙이다. 예를 들어, 뷰포트의 너비가 (1) 400px 미만일 때, (2) 400px 이상 800px 미만일 때, (3) 800px 이상 1200px 미만일 때, (4) 1200px 이상일 때의 디자인을 각각 다르게 설계하는 상황을 가정해보자. 먼저 (1)의 디자인을 먼저 작성하고, (2)의 디자인 중 (1)과 다른 부분을 `@media query` 구문을 이용하여 400px 이상의 뷰포트에 대해 작성한다. 이렇게 작성하면 (1)에서 작성한 디자인 중 (2)에 의해 덮어씌워지지 않는 디자인은 그대로 유지된다. (3), (4)도 마찬가지의 방법으로 작성하여 완성한다.

3.7 CSS Exercises

Exercise 1

Code 3.19가 **Figure 3.6**과 같이 나타나도록 style.3.1.css를 작성하여라.

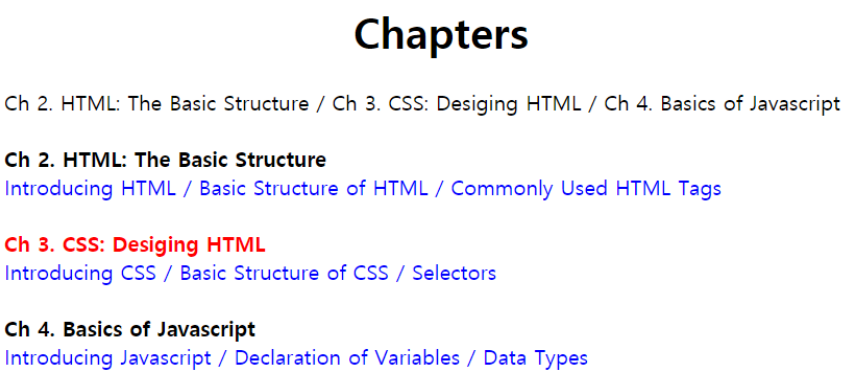


Figure 3.6 Exercise 1 Example

Code 3.19 Exercise 1

```
<!doctype html>
<html>
<head>
    <title>CSS Exercise 1</title>
    <meta charset="utf-8">
    <link rel="stylesheet" text="type/css" href="./style.3.1.css">
</head>
<body>
    <h1>Chapters</h1>
    <div class="horizontal-lists">
        Ch 2. HTML: The Basic Structure / Ch 3. CSS: Desiging HTML
        / Ch 4. Basics of Javascript
    </div><br>
    <div class="chapter">
        <div class="chapter-name">Ch 2. HTML: The Basic Structure</div>
        <div class="horizontal-lists">
            Introducing HTML / Basic Structure of HTML / Commonly Used HTML Tags
        </div>
    </div><br>
    <div class="chapter" id="current-chapter">
        <div class="chapter-name">Ch 3. CSS: Desiging HTML</div>
        <div class="horizontal-lists">
            Introducing CSS / Basic Structure of CSS / Selectors
        </ul>
    </div><br>
    <div class="chapter">
        <div class="chapter-name">Ch 4. Basics of Javascript</div>
        <div class="horizontal-lists">
            Introducing Javascript / Declaration of Variables / Data Types
        </div>
    </div>
</body>
</html>
```

Exercise 2

Code 3.20이 **Figure 3.7**과 같이 나타나도록 style.3.2.css를 작성하여라.

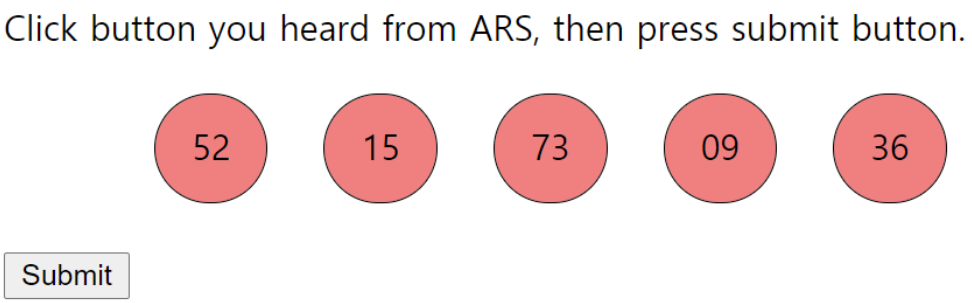


Figure 3.7 Exercise 2 Example

Code 3.20 Exercise 2

```
<!doctype html>
<html>
<head>
    <title>CSS Exercise 2</title>
    <meta charset="utf-8">
    <link rel="stylesheet" text="type/css" href="./style.3.2.css">
</head>
<body>
    <div class="container" style="width: 500px">
        <div class="header">
            Click button you heard from ARS, then press submit button.
        </div>
        <div class="numbers">
            <div class="number">52</div>
            <div class="number">15</div>
            <div class="number">73</div>
            <div class="number">09</div>
            <div class="number">36</div>
        </div>
        <div class="footer">
            <button type="submit">Submit</button>
        </div>
    </div>
</body>
</html>
```

Exercise 3

Code 3.21을 참고하여, **Figure 3.8**과 같이 우측 하단에 고정되어 있으면서, 클릭하였을 때 웹 페이지의 제일 상단과 제일 하단으로 이동하는 버튼을 `#scroll-button` 내부를 수정하고, style.3.3.css를 작성하여 구현하여라. a 태그의 `href` 속성값을 HTML 요소의 id로 설정하면 해당 위치로 이동할 수 있으며, 위쪽 삼각형(▲)과 아래쪽 삼각형(▼)의 개체 번호는 각각 9650, 9660이다.

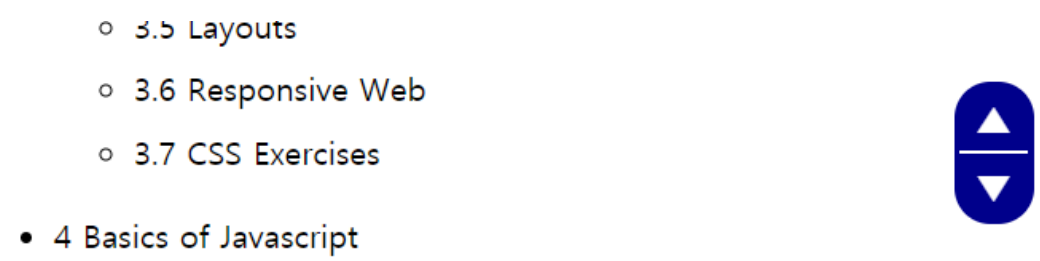


Figure 3.8 Exercise 3 Example

Code 3.21 Exercise 3

```
<!doctype html>
<html>
<head>
    <title>CSS Exercise 3</title>
    <meta charset="utf-8">
    <link rel="stylesheet" text="type/css" href="./style.3.3.css">
    <style> #content { padding: 30px 100px; margin: 0 auto }
        #scroll-button a { text-decoration: none; }
        ul { margin: 20px 0 } li { margin: 10px 0 } </style>
</head>
<body>
    <div id="page-top"></div>
    <div id="content"><ul>
        <li>1 Introduction to Front-end</li>
        <ul><li>1.1 Introducing WEB</li><li>1.2 2021 Study Schedule</li>
            <li>1.3 Setting Development Environment</li></ul>
        <li>2 HTML: The Basic Structure</li>
        <ul><li>2.1 Introducing HTML</li><li>2.2 Basic Structure of HTML</li>
            <li>2.3 Commonly Used HTML Tags</li><li>2.4 Class and Id Attributes</li>
            <li>2.5 HTML Exercises</li></ul>
        <li>3 CSS: Desinging HTML</li>
        <ul><li>3.1 Introducing CSS</li><li>3.2 Basic Structure of CSS</li>
            <li>3.3 Style Properties</li><li>3.4 Selectors</li> <li>3.5 Layouts</li>
            <li>3.6 Responsive Web</li><li>3.7 CSS Exercises</li></ul>
        <li>4 Basics of Javascript</li>
        <ul><li>4.1 Introducing Javascript</li><li>4.2 Declaration of Variables</li>
            <li>4.3 Data Types</li><li>4.4 Statements and Functions</li>
            <li>4.5 Built-in Objects</li><li>4.6 Basics of JS Exercises</li></ul>
        <li>5 Javascript: Dynamic Frontend</li>
        <ul><li>5.1 Javscript with Front-end</li> <li>5.2 Document Object Model</li>
            <li>5.3 Browser Object Model</li><li>5.4 Event and Event Listener</li>
            <li>5.5 JS Exercises</li></ul>
        <li>A Exercise Answers</li>
        <ul><li>A.1 HTML Exercise Answers</li><li>A.2 CSS Exercise Answers</li>
            <li>A.3 Basics of Javascript Exercise Answers</li>
            <li>A.4 JS Exercise Answers</li></ul>
    </ul></div>
    <div id="page-bottom"></div>
    <div id="scroll-button">
        <!-- Your code here -->
    </div>
</body>
</html>
```

Exercise 4

Code 3.22를 참고하여 style.3.4.css를 작성하여라. `body` 태그의 배경색이 600px 미만일 때는 `skyblue`, 600px 이상 1200px 미만일 때는 `blue`, 1200px 이상일 때는 `darkblue`이어야 하며, `h1` 태그의 색은 900px 미만일 때는 `white`, 900px 이상일 때는 `yellow`이어야 한다.

Code 3.22 Exercise 4

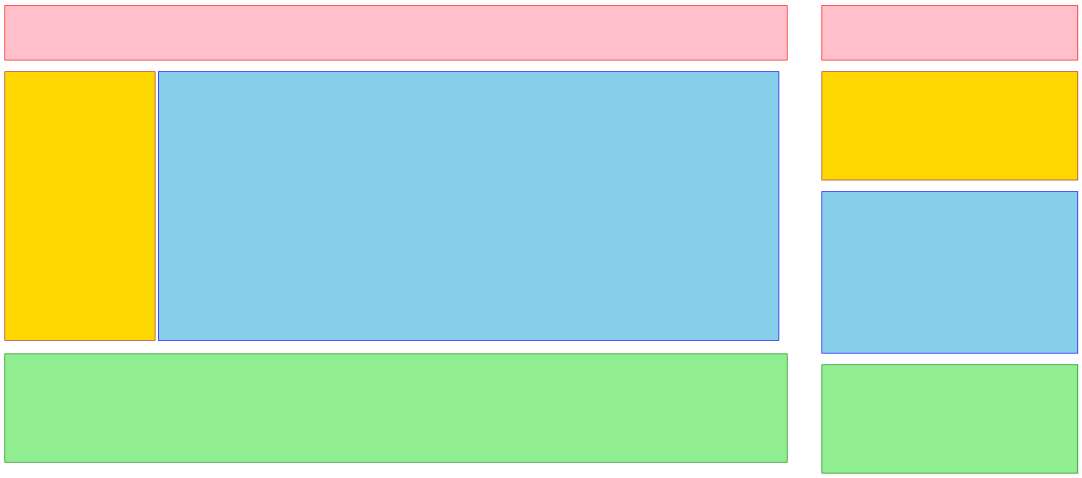
```
<!doctype html>
<html>
<head>
    <title>CSS Exercise 4</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1" >
    <link rel="stylesheet" text="type/css" href="./style.3.4.css">
</head>
<body>
    <h1>Responsive Web</h1>
</body>
</html>
```

Exercise 5

Code 3.23을 참고하여 웹 페이지의 너비가 800px 이상이면 **Figure 3.9a**, 800px 미만이면 **Figure 3.9b**와 같이 렌더링 되도록 style.3.5.css 파일을 작성하여라.

Code 3.23 Exercise 5

```
<!doctype html>
<html>
<head>
    <title>CSS Exercise 5</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1" >
    <link rel="stylesheet" text="type/css" href="./style.3.5.css">
</head>
<body>
    <div class="header"></div>
    <div class="body">
        <div class="navbar"></div>
        <div class="content"></div>
    </div>
    <div class="footer"></div>
</body>
</html>
```



(a) Wider than 800px **(b)** Smaller than 800px

Figure 3.9 Webpage view of Exercise

Chapter 4

Basics of Javascript

Contents

4.1 Introducing Javascript

자바스크립트(Javascript; JS)는 웹 분야에서 광범위하게 사용되는 인터프리터형 프로그래밍 언어이다. Front-end에서는 JS를 이용하여 웹 페이지의 동적인 기능을 구현하며, 프론트엔드 프레임워크인 Angular.js와 React.js 등에서도 쓰인다. 또한, front-end 뿐만 아니라 back-end 애플리케이션 역시 Node.js 엔진을 이용하여 JS로 구현할 수 있다. 이러한 JS의 특징으로 인해 2021년 기준 Stackoverflow에서 조사한 컴퓨터 언어의 사용 빈도 순위에서 7년째 1위를 차지할 정도로 여러 분야에서 광범위하게 응용되고 있다.

컴파일형 언어와 인터프리터형 언어

컴퓨터 언어는 그 언어로 작성된 프로그램의 실행 방식에 따라 크게 컴파일러형 언어와 인터프리터형 언어로 나뉜다. 컴파일러형 언어로 작성된 프로그램은 실행하기 위해 소스 코드를 기계어로 변환하는 과정, 즉 컴파일 과정이 필요하며, C와 C++, Java, Rust, Go 등이 대표적인 컴파일러형 언어이다.

반면 인터프리터형 언어는 컴파일 과정 없이 인터프리터를 통해 소스 코드를 바로 실행할 수 있는 언어로, Python과 JS, Lisp 계열 언어들이 대표적인 인터프리터형 언어이다. 인터프리터형 언어들은 컴파일 과정이 불필요하므로 인터프리터에 하나의 표현식(expression)을 입력한 후 바로 그 코드의 실행 결과를 확인할 수 있으며, REPL(Read-Eval-Print-Loop) Shell에서 하나의 표현식을 실행한 후 결과를 바로 확인할 수 있다.[1] 그러나 인터프리터형 언어는 표현식을 분석하고 컴파일하는 과정을 해당 표현식이 실행될 때마다 반복해야 하므로 컴파일형 언어에 비해 성능은 낮은 편이다.

Code 4.1 Simple Example of JS

```
> console.log('Hello World!');
undefined
Hello World!
> alert('Hello World!');
undefined
> 3 + 5;
8
```

웹 브라우저 개발자 도구의 Console 탭은 JS의 REPL Shell 역할을 하므로, Console 탭에서 **Code 4.1**을 실행하여 결과를 관찰해보자.

첫 번째 표현식을 실행하면 콘솔에 `undefined`와 `Hello World!`라는 문자열이 출력되는데, 먼저 표현식의 반환값인 `undefined`가 console에 출력되고, 이 표현식이 `Hello World!`라는 문자열을 출력하라는 뜻을 가지므로 해당 문자열 역시 console에 출력된다. 두 번째 표현식을 실행하면 브라우저에 `Hello World!` 문자열을 담은 경고창이 나타나며, 역시 해당 표현식의 반환값인 `undefined`가 console에 출력된다. 또한, 마지막 표현식을 실행하면 3과 5의 덧셈 결과값이 출력된다. 앞으로 본 교재에서는 표현식의 반환값이 `undefined`인 경우에는 결과 표시를 생략한다.

[1] 파일 형태로 작성하는 것도 당연히 가능하다. JS 파일 작성과 실행은 5장에서 다룬다.

4.2 Declaration of Variables

ECMAScript

ECMAScript는 스크립트 언어에 관한 규약으로, 줄여서 ES라고 하며, JS는 ES의 표준을 따른다. ES1이 1997년 출시된 이후 2015년 ES6가 출시되며 언어의 표준에 큰 변화가 발생하였고, 이후 지속적인 업데이트를 통해 2019년 ES9가 출시되어 현재[2]에 이르고 있다. 본 교재에서는 ES9을 표준으로 한 JS를 다룬다.

식별자의 선언

JS는 다른 언어들과 마찬가지로 변수나 상수 등의 식별자(identifier)를 정의하고 사용할 수 있고, 모든 식별자에는 자료형이 있다. 다만 C나 Java 등과는 달리 식별자를 선언할 때 자료형을 명시해주지 않고, 인터프리터가 식별자에 할당되는 값을 분석하여 자료형을 스스로 지정한다.

변수, 상수, 함수 등은 camel case(예: `parsedSourceInput`), 클래스는 pascal case(예: `RequestHandler`), 상수는 대문자와 underscore(_)로 작명(예: `LENGTH_LIMIT`)한다. 또한, 숫자로 시작하지 않고 가급적 알파벳으로 시작하며, 식별자가 나타내는 바를 명확하게 알 수 있도록 작명한다.

식별자는 `var`, `let`, `const` 등의 키워드를 이용하여 선언한다. 먼저, `var`는 지금까지 널리 사용되어 온 키워드로, `var`로 선언한 변수는 재선언할 수 있고, 값을 재할당할 수도 있다.

Code 4.2 Declaring Variable using `var`

```
> var num1 = 3;
> num1;
3
> num1 + 5;
8
> num1 = 5;
5
> var num1 = 20;
```

반면 `let`으로 선언한 변수는 값을 재할당할 수는 있으나, 재선언은 불가능하다.[3]

Code 4.3 Declaring Variable using `let`

```
> let num2 = 3;
> num2 + 5;
8
> num2 = 5;
5
> let num2 = 20;
SyntaxError: redeclaration of let num2
```

[2] 2021년
[3] Chrome의 개발자 도구를 이용하였을 때 재선언이 가능한 버그가 보고되었다. (2021년 기준)

const는 constant, 즉 상수(常數)의 약자로, const로 선언한 상수는 재선언과 재할당이 모두 불가능하다.

Code 4.4 Declaring Constant using const

```
> const constNum = 3;
undefined
> constNum + 5;
8
> constNum = 5;
TypeError: invalid assignment to const 'constNum'
> const constNum = 20;
Thrown:
SyntaxError: redeclaration of const constNum
```

var는 JS 초창기부터 사용되어 왔던 식별자 선언 키워드인 반면, let과 const는 ES6 출시와 함께 추가된 식별자 선언 키워드이다. var과 let/const는 재선언 가능성에서도 차이점이 있지만, 식별자에 접근할 수 있는 범위(scope)에도 차이가 있다.

Code 4.5 Scope of var and let

```
> { var num1 = 1; }
> num1;
1
> { let num2 = 2; }
> num2;
Uncaught ReferenceError: num2 is not defined
```

Code 4.5를 실행하면, var로 선언된 num1의 값은 정상적으로 출력되나, let으로 선언된 num2는 정의되지 않았다는 에러가 발생한다. 이렇듯 var로 선언된 식별자는 선언된 블록 바깥에서 접근될 수 있고 함수 밖에서는 접근될 수 없는 function-scope이며, let과 const로 선언된 식별자는 선언된 블록 바깥에서 접근될 수 없는 block-scope이다.

Table 4.1 Differences between var, let and const

키워드	재할당 (Reassignment)	재선언 (Redeclaration)	스코프 (Scope)
var	가능	가능	function-scope
let	가능	불가능	block-scope
const	불가능	불가능	block-scope

ES6 이전에는 var를 이용하여 변수를 선언하였으나, ES6 이후에는 let과 const가 재선언이 불가능하고, block-scope이기 때문에 var의 사용은 지양되고 let과 const를 이용하여 변수와 상수를 선언하는 것이 권장된다. var는 function-scope이기 때문에 코드의 네임스페이스를 오염[4]시키며, run-time에서 의도치 않은 논리적 오류를 발생시킬 수 있다. 반면 let과 const로 선언된 식별자는 block-scope이므로 네임스페이스 오염을 최소화할 수 있고, const로 선언된 상수는 재할당이 불가능하므로 run-time에서 상수값이 변경되어 발생하는 논리적 오류를 방지할 수 있다.

[4]https://stackoverflow.com/questions/22903542/what-is-namespace-pollution

4.3 Data Types

JS에는 다음과 같은 7개의 자료형이 존재하며, 이번 절에서는 symbol형을 제외한 6가지 자료형과 배열을 다룬다.

- Primitive Types: number, string, boolean, undefined, null, symbol
- Compound Type: object

Number

number형은 수(數)를 저장하는 자료형으로, 정수, 유리수 등을 구별 없이 저장할 수 있고, 상호 연산이 가능하다.

Code 4.6 Number Type

```
> const num1 = 2;
> const num2 = 7.5;
> const num3 = 2 / 5;
> typeof num1;
'number'
> typeof num2;
'number'
> num1 * num2;
15
> num3;
0.4
```

number 자료형에는 일반적인 실수 이외에 `Infinity`와 `NaN`(not a number)이 있다. 두 값은 0으로 나누는 연산이 수행되면 발생한다.

Code 4.7 Infinity and NaN

```
> 3 / 0;
Infinity
> typeof(3 / 0);
'number'
> 3 / (-0);
-Infinity
> 0 / 0;
NaN
> typeof(0 / 0);
'number'
```

ES7부터 지수 연산자를 지원하며, **Code 4.8**과 같이 사용한다.

Code 4.8 Exponentation Operator

```
> 3 ** 5
243
> 4 ** (-0.5)
0.5
```

String

string형은 문자열을 저장하는 자료형으로, string형 문자열 간에도 상호 계산이 가능하다.

Code 4.9 String Type

```
> const hello = 'Hello';
> const world = 'World';
> const helloWorld = hello + ' ' + world;
> helloWorld;
'Hello World'
> helloWorld.length;
11
> helloWorld[4];
'o'
> typeof helloWorld;
'string'
```

문자열을 표현할 때 큰따옴표(")나 작은따옴표(')를 사용하여 문자열을 감싸며, 서로 다른 문자로 감싸는 것은 불가능하다. 또한, 문자열 내에 감싸는 문자 등이 포함되어 있는 경우 역슬래시(\)를 이용하여 escape 시켜야 한다.

Code 4.10 Assigning Value on String Type Variable

```
> "String";
'String'
> 'String';
'String'
> "String';
Uncaught SyntaxError: Invalid or unexpected token
> "McDonald's";
'McDonald\'s'
> console.log("McDonald's");
McDonald's
```

Boolean

boolean형은 논리적인 요소를 나타내는 자료형으로, `true`와 `false` 두 가지의 값이 가능하다.

Code 4.11 Boolean Type

```
> 1 == 1;
true
> 1 > 3;
false
> true == false;
false
> true != false;
true
> const isNumberEven = (5 % 2 == 0);
> isNumberEven;
false
```

Object

object형, 즉 객체형이란 key와 value를 가질 수 있는 자료형이다. 예를 들어, 어떠한 학생에 대한 정보를 다룰 때, 학생의 학번, 이름, 학과, 재학 여부 등을 서로 다른 변수에 저장하지 않고, `student`라는 객체형 식별자에 저장하여 일괄적으로 다룰 수 있다. Key는 같은 계층에서 유일한 문자열이고, value에는 숫자, 문자열 등 자료형에 상관없이 모든 종류의 값이 할당될 수 있다. 즉, 객체 내부에 또 다른 객체가 포함될 수 있어, 계층이 있는 데이터(hierarchial data)를 형성하는 것이 가능하다.

Code 4.12 Object Type

```
> const person = {
    age: 21,
    name: 'Frank',
    height: 170,
    isMale: true,
};
> person;
{ age: 21, name: 'Frank', height: 170, isMale: true }
> person.age;
21
> person['age'];
21
> !person.isMale;
false
> person.weight = 60;
60
> person;
{ age: 21, name: 'Frank', height: 171, isMale: true, weight: 60 }
```

Undefined and Null

undefined형과 null형은 각각 값이 할당되지 않은 식별자와 값을 모르는 식별자가 갖는 자료형이며, 두 자료형의 값은 각각 항상 `undefined`와 `null`이다.

Code 4.13 Undefined Type

```
> let undefVar;
> undefVar;
undefined
> typeof undefVar;
'undefined'
```

JS에는 null 자료형 식별자를 `typeof` 함수를 이용하여 확인할 때 잘못된 자료형을 반환받는 버그를 가지고 있다. 이는 매우 오래된 버그이므로 고치는 것이 불가능하다고 하며, 변수가 `null`임을 확인하기 위해서는 값을 직접 비교하여야 한다.

Code 4.14 Null Type

```
> const nullVar = null;
> typeof null;
'object'
> nullVar == null;
true
```

Array

배열은 index를 가지는, 복수의 자료를 저장할 수 있는 자료구조이다. JS에서는 각 원소의 자료형에 제한이 없어 서로 다른 자료형의 원소를 하나의 배열에 저장할 수 있으며, 배열의 크기가 정해져 있지 않아 각종 메서드(method)를 이용하여 원소를 자유롭게 넣고 뺄 수 있다.

Code 4.15 Array

```
> const arr = [0, 1, 1.5, true, 'hungry'];
> arr.length;
5
> arr[3];
true
> arr[4];
'hungry'
> arr.push('new element');
6
> arr;
[ 0, 1, 1.5, true, 'hungry', 'new element' ]
```

4.4 Statements and Functions

Comparison

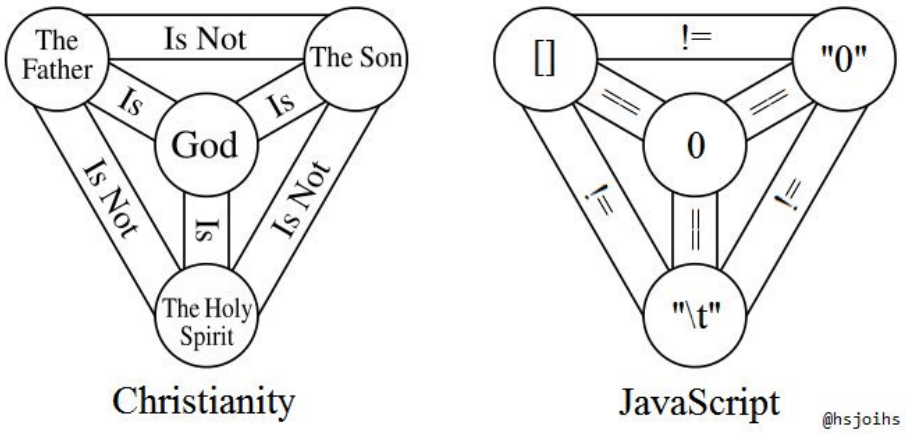
JS의 equality comparison은 여타 언어들과 유사하게 ==과 != 연산자로 수행될 수 있으며, 비교 연산의 결과가 참이면 `true`, 거짓이면 `false`를 반환한다. 그러나 JS에서는 ==이나 != 연산자로 비교 연산을 수행하였을 때 문제가 발생할 수 있다.

Code 4.16 Loose Equality

```
> 0 == 0;
true
> 0 == '0';
true
```

Code 4.16은 number형의 0과 string형의 '0'을 비교하는 코드인데, 두 값은 자료형이 다름에도 불구하고 같다는 결과가 도출된다. 그 이유는 JS에서 ==과 != 연산자를 이용하여 비교 연산을 수행할 때, 두 값에 대해 자의적으로 자료형 변환(type coercion)을 수행한 뒤 변환된 값을 비교하기 때문이다. 이렇게 비교 연산에서 ==과 != 연산자를 사용하여 자료형 변환이 이루어진 뒤 수행되는 연산을 loose equality이라고 한다. JS가 수행하는 타입 변환 규칙은 복잡하기 때문에 loose equality는 사용하지 않는 것이 권장된다.

(a) Syllogism does not hold



(b) The Javascript Trinity of 0

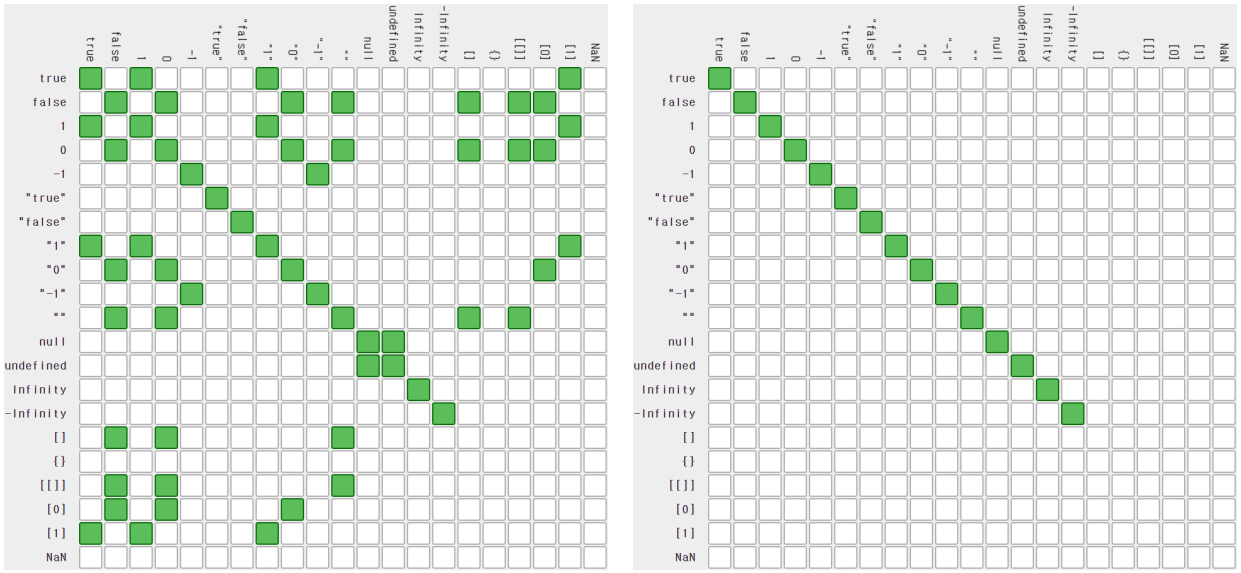
Figure 4.1 Loose Equality Memes

JS에서는 loose equality 대신 자료형까지 고려하여 값을 비교하는 **strict equality**가 권장된다. Strict equality는 ===과 !== 연산자를 이용하여 수행된다.

Code 4.17 Strict Equality

```
> 0 === 0;
true
> 0 === '0';
false
```

Figure 4.2는 loose equality와 strict equality의 결과를 정리한 표이다. 이 결과를 알아두어서 나쁠 것은 없지만, 직관적이고 논리적 오류를 최소화할 수 있는 strict equality를 사용하는 것이 훨씬 효율적이다.



(a) Loose Equality

(b) Strict Equality

Figure 4.2 Equality table of JS. Colored cell implies `true`[5]

MDN 문서[6]에서 equality comparison에 대해 더 자세한 내용을 확인할 수 있다.

Conditional Statements

JS의 조건문은 여타 언어들의 조건문과 유사하게 `if`, `if else`, `else`의 키워드를 사용한다.

Code 4.18 If-else Statement

```
> const num = 100;
> const isEven = num % 2 === 0;
> if (isEven) {
      console.log('num is even');
  } else {
      console.log('num is odd');
  }
num is even
> const remainder = num % 3;
> if (remainder === 0) {
      console.log('remainder is 0');
  } else if (remainder === 1) {
      console.log('remainder is 1');
  } else {
      console.log('remainder is 2');
  }
remainder is 1
```

[5]https://dorey.github.io/JavaScript-Equality-Table/
[6]https://developer.mozilla.org/en-US/docs/Web/JavaScript/Equality_comparisons_and_sameness

또한, switch-case문과 삼항 연산자 ? : 도 지원한다.

Code 4.19 switch-case Statement and Ternary operator

```
> const num = 100;
> const remainder = num % 3;
> switch (remainder) {
      case 1:
          console.log('remainder is 1'); break;
      case 2:
          console.log('remainder is 2'); break;
      default:
          console.log('remainder is 0');
  }
remainder is 1
> const quotient = ((num % 7 === 0) ? num : (num - num % 7)) / 7;
14
```

Iteration Statements

JS의 반복문 역시 여타 언어들과 유사하게 for문을 사용하며, while문과 do-while문 역시 지원한다.

Code 4.20 for Statement

```
> const fruits = ['apple', 'orange', 'mango', 'grapes'];
> for (let i = 0; i < fruits.length; i++) {
      console.log(fruits[i]);
  }
apple
orange
mango
grapes
```

Code 4.20과 같이 반복문을 사용할 때마다 배열의 index를 어떠한 변수에 저장하고, 이를 이용해 배열의 각 원소를 가져오는 것은 번거롭다. for-of문을 사용하여 반복문을 간결하게 사용할 수 있다.

Code 4.21 for-of Statement

```
> const fruits = ['apple', 'orange', 'mango', 'grapes'];
> for (const fruit of fruits) {
    console.log(fruit);
  }
apple
orange
mango
grapes
```

Code 4.21과 같이 for-of문은 배열의 각 원소를 차례대로 특정 변수(`fruit`)에 저장하고, for문 내에서 사용할 수 있도록 한다. 이렇게 for-of문을 사용하면 각 원소의 index는 알 수 없지만, 개별 원소는 직접 가져와 반복문을 간결하게 작성할 수 있다는 장점이 있다.

Code 4.22 for-in Statement

```
> const person = { age: 21, name: 'Frank', height: 171, isMale: true, weight: 60 };
> for (const key in person) {
      console.log(key + ': ' + person[key]);
  }
age: 21
name: Frank
height: 171
isMale: true
weight: 60
```

for-of문과 유사하여 헷갈리기 쉬운 for-in문은 객체(object)에 대해 동작하며, for-in문 내부에서 객체의 각 key를 직접 가져와 사용할 수 있고, 이를 이용해 value에도 바로 접근할 수 있다.

Functions

Code 4.23 Function Declaration and Call

```
> function add(x, y) {
      return x + y;
  }
> add(1, 2);
3
> add('1', '2');
'12'
```

JS에서 함수는 **Code 4.23**과 같이 `function` 키워드를 이용하여 함수의 이름과 인자를 작성하고, 중괄호 내부에 함수 내용을 작성한다. JS에서는 식별자의 자료형을 따로 지정하지 않으므로 인자와 반환값의 자료형 역시 따로 지정하지 않는다는 점을 주목하여야 한다. 따라서, **Code 4.23**의 함수는 마지막 줄과 같이 사용될 수도 있다.

Code 4.24 Arrow Function

```
> const add = (x, y) => {
      return x + y;
  };
> add(1, 2);
3
> const square = num => num * num;
> square(5);
25
```

ES6 이후에는 **Code 4.24**와 같이 arrow(`=>`)를 사용하여 함수를 비교적 간결하게 정의할 수 있는 **arrow function**이 권장된다. 표현식을 관찰해보면, `x`와 `y` 두 인자를 받아 덧셈을 수행한 뒤 반환하는 함수를 `add`라는 식별자에 할당하였다. JS에서는 함수 역시 하나의 변수 또는 상수로 취급되어 arrow function을 이용하여 함수를 작성할 수 있고, 함수를 다른 함수의 인자로 넘길 수도 있다.

Arrow function의 인자가 1개일 때는 인자 부분의 괄호를 생략할 수 있고, 함수에 값을 반환하는 부분만 있는 경우 중괄호와 `return` 키워드가 생략될 수 있다.

Code 4.25 Anonymous Function

```
> num => num * num;
function (num)
> ((base, power) => {
      res = 1;
      for (let i = 0; i < power; i++) res *= base;
      return res;
  })(3, 5);
243
```

Arrow function을 활용하면 **Code 4.25**와 같이 익명 함수(anonymous function)를 선언할 수 있다.

Code 4.26 Anonymous Function with High-order Function

```
> const arr = ['a', 'bb', 'ccc', 'dddd'];
> const lengths = arr.map(e => e.length);
> lengths;
[ 1, 2, 3, 4 ]
```

함수 중에는 함수를 인자로 쓰는 함수가 있는데, 이러한 함수를 고차원 함수(high-order function)라고 한다. 익명 함수는 재사용하지 않는 함수를 고차원 함수의 인자로 넘길 때 유용하게 사용된다. **Code 4.26**에서 `map` 함수는 배열의 각 원소를 인자로 받은 함수의 인자로 주어 실행한 결과를 배열로 반환한다. JS에서는 이러한 고차원 함수가 특히 많으므로 고차원 함수를 사용하면서 익명 함수를 빈번하게 사용하게 될 것이다.

Code 4.27 `forEach` Method

```
> const fruits = ['apple', 'orange', 'mango', 'grapes'];
> fruits.forEach(fruit => console.log(fruit));
apple
orange
mango
grapes
> const printFruit = (fruit, idx) => console.log('#' + (idx + 1) + ': ' + fruit);
> fruits.forEach(printFruit);
#1: apple
#2: orange
#3: mango
#4: grapes
```

익명 함수를 이용하여, 앞에서 다룬 for-of 문을 **Code 4.27**과 같이 `forEach` 함수를 사용하여 바꿀 수 있다. `forEach` 함수는 각 원소뿐만 아니라 원소의 index와 원본 배열까지도 받아 사용할 수 있다.

4.5 Built-in Objects

이번 절에서는 4.3절에서 다룬 객체에 대해 더 자세히 알아본다. JS에서 number형이나 string형의 식별자, 배열, 함수 등은 모두 객체[7]이고, 그러므로 객체라는 개념을 정확히 이해하기 위해 객체 지향형 프로그래밍(Object-Oriented Programming)을 공부해야 하지만, 본 교재에서 다루는 내용과는 다소 거리가 있으므로 이번 절에서는 클래스와 메서드 등의 기초 개념만 다룬다. 또한, JS에서 기본적으로 제공되는 객체(built-in objects)의 종류와 속성과 메서드 등의 사용 방법에 대해 학습한다.

Property와 Method

Code 4.12(48쪽)에서 `person`이라는 객체에 `age`, `name`, `height`, `isMale`의 네 속성과 그 값을 부여하였다. 이때 `person` 객체의 속성의 값을 이용하여 수행하는 작업을 생각해보자. 예를 들어 새해가 되어 나이를 증가시켜주는 작업은 `age`의 값을 1만큼 증가시키는 작업이다. 이 작업은 매우 간단하지만, 복잡하고 긴 과정을 거쳐야 하는 작업일 수도 있으므로 `increaseAge`라는 함수를 만들어 사용하는 상황을 가정하자.

JS 파일에 이러한 함수를 만들어두고 사용하면 어떤 부작용이 있겠는가? 객체가 많은 프로그램의 경우는 함수의 수가 지나치게 많아져 가독성이 저하되고, 개발자가 헷갈리기 쉬워지고 코드 분석도 어려워지는 등 효율성이 저하될 수 있다. 또한, 다른 객체에 대해 유사한 작업을 수행하는 함수가 있다면 함수 이름이 겹쳐 namespace 오염이 일어날 수 있고, 함수 이름을 비효율적으로 짓게 된다. 따라서 객체의 속성값을 이용하는 함수는 객체 내부에 정의하여 namespace 오염과 부작용을 방지하고, 이를 **메서드(method)**라고 한다.

Code 4.28 Method of Object

```
> const person = {
      age: 21,
      name: 'Frank',
      height: 170,
      isMale: true,
      increaseAge: function () { this.age++ },
  };
> person.increaseAge();
> person.age;
22
```

메서드는 **Code 4.28**과 같이 key에 메서드 이름을 작성하고 value에 메서드가 수행할 작업을 함수의 형태로 할당한다. 앞의 `increaseAge` 메서드는 자신이 속한 `person` 객체의 `age` 값을 1만큼 증가시켜야 하는데, 함수 내에 `age`가 선언되어 있지 않아 직접 접근할 수 없다. 이때 `this` 키워드를 사용하면 자기 자신이 속한 객체에 접근할 수 있다.[8] 즉 `increaseAge` 메서드 내부의 `this`는 `person`을 가리키고, `increaseAge`는 자신의 속성 `age`의 값을 1만큼 증가시킨다.

[7] object형은 아니다.
[8] `this`는 자기 자신의 객체 외에도 다른 종류의 객체를 가리킬 수도 있다.

Code 4.29 Simple Method Syntax

```
> const person = {
      age: 21,
      increaseAge() { this.age++ },
  };
> person.increaseAge();
> person.age;
22
```

ES6의 축약 메서드 표현법에 따라 **Code 4.29**와 같이 축약하여 표현할 수 있다.

메서드를 표현할 때 arrow function을 이용하면 오류가 발생할 여지가 크다는 점을 주의해야 한다. 화살표 함수는 `this`가 가리키는 객체를 정하는 방식이 일반적인 함수와 다르므로[9] 메서드를 표현할 때는 arrow function보다는 전통적인 함수 표현 방식을 사용하는 것이 좋다.

Class and Instance

앞에서는 Frank라는 이름을 가진 사람에 대한 정보를 다루기 위해 `person`이라는 객체를 생성하고 속성과 메서드를 정의하였다. 그런데 Frank라는 사람 한 명뿐만 아니라 같은 속성을 갖는 수많은 사람에 대한 정보를 다룰 때 **Code 4.28**과 같이 객체를 일일이 생성하는 방식은 매우 비효율적이며 재사용성이 크게 떨어진다.

이렇게 동일한 형태의 객체들을 생성하기 위해 **클래스(class)**를 이용하여 객체들의 구조를 표현하며, 클래스는 설계도와 같은 역할을 한다. 클래스는 각 객체가 갖는 속성, 상수, 메서드 등에 대한 정보의 집합이다. 그리고 이러한 설계도, 즉 클래스에 따라 생성된 각 객체를 **인스턴스(instance)**라고 한다. 예를 들어 `Person`이라는 클래스가 있다면 이 클래스에 따라 생성된 객체 `frank`, `david`, `jennie` 등은 인스턴스이다.

인스턴스를 생성하려면 클래스라는 설계도를 이용하여 객체를 생성하는 매개체가 필요하다. 이 매개체를 **생성자(constructor)**라고 부르며, 생성자는 인스턴스를 생성할 때 필요한 정보를 받아, 설계도에 따라 객체의 속성이나 메서드를 정한다. 예를 들어, 앞의 `Person` 클래스의 생성자는 나이, 이름, 키, 성별 등의 정보를 받아 `age`, `name`, `height`, `isMale`의 속성값에 저장할 것이다. 또한, 생성자는 여러 유의미한 정보가 응집되어 있는 정보를 나누어 서로 다른 속성에 저장하기도 한다. 예를 들어, 학생의 학번을 받아서 입학 연도, 소속 학과, 개인 일련번호 등의 정보를 저장하는 `Student`라는 클래스를 가정하면 이 클래스의 생성자는 학번을 분석하여 `admissionYear`, `department`, `serialNumber` 등의 속성에 분석한 값들을 저장할 수 있다.

본 교재에서는 생성자를 직접 작성하는 것은 다루지 않고, 생성자를 이용하여 인스턴스를 생성할 때는 `new` 키워드를 사용한다. 예를 들어 `Person` 클래스는 `age`, `name`, `height`, `isMale` 값을 인자로 받아 인스턴스를 생성한다면 `jennie`라는 인스턴스는 `jennie = new Person(24, 'Jennie', 163, false);`와 같이 생성할 수 있다. 또한, 어떤 속성이나 메서드는 생성자에 의해 할당되는 속성이나 메서드가 필요하지 않을 수 있는데, 이러한 메서드를 정적(static) 속성 및 메서드라고 하며, 생성자를 사용하지 않고 클래스에서 직접 속성이나 메서드를 호출할 수 있다.

여담으로, 객체는 `const` 키워드로 선언되어도 그 속성이나 메서드는 재할당될 수 있다.

[9] https://poiemaweb.com/es6-arrow-function

Global Methods(Functions)

다음 메서드들은 객체를 명시하지 않고 함수 형태로 사용할 수 있는 메서드들이다. 이 함수들은 사실 전역 객체(global object)의 메서드이므로 함수처럼 사용할 수 있는 것이다.

- `isFinite(testValue) => boolean`: 주어진 값이 유한한지 판별하는 메서드; `testValue`를 인자로 받아 `Infinity`, `NaN`, `undefined`이면 `false`, 아니면 `true`를 반환한다.
- `isNaN(testValue) => boolean`: 주어진 값이 `NaN`인지 판별하는 메서드; `testValue`를 인자로 받아 `NaN`이면 `false`, 아니면 `true`를 반환한다.
- `parseInt(string[, radix]) => number`: 주어진 문자열을 숫자로 변환하여 반환하는 메서드; `string`과 `radix`를 인자로 받아, `string`을 `radix`진법의 정수로 변환하여 반환한다. 주어진 값을 정수로 변환할 수 없다면 `NaN`을 반환하며, `radix`가 주어지지 않을 때는 `string`이 `0x`나 `0X`로 시작하면 16진수로, 그 외에는 10진수로 변환한다.

Code 4.30 Global Methods

```
> isFinite(3/0);
false
> isNaN(0/0);
true
> parseInt('3.5');
3
> parseInt('hello');
NaN
```

String Objects

string형 식별자, 즉 `String` 객체의 자주 쓰이는 속성이나 메서드이며, MDN 문서[10]에서 더 자세하게 확인할 수 있다.

- `length`: 문자열의 길이를 값으로 갖는 속성
- `includes(searchString[, position]) => boolean`: 문자열에 주어진 문자열이 포함되는지 판별하는 메서드; 원래 문자열에 `searchString` 문자열이 `position`번째 문자 이후에 포함되면 `true`, 포함되지 않으면 `false`를 반환한다. `position`의 기본값은 `0`이다.
- `replace(substr, newSubstr) => string`: 문자열의 일부분을 다른 문자열로 대치(代置)하여 반환하는 메서드; 원래 문자열에서 처음으로 나오는 `substr` 문자열 부분을 `newSubstr`로 대치한다. `substr`은 정규식 형태로 주어질 수도 있다.
- `replaceAll(substr, newSubstr) => string`: 문자열의 일부분을 모두 다른 문자열로 대치하여 반환하는 메서드; `replace` 메서드와 유사하게 동작하나 `substr` 문자열 부분을 모두 `newSubstr`로 대치한다.
- `split([separator, limit]) => Array<string>`: 문자열을 주어진 문자열을 기준으로 분리하여 문자열의 배열로 반환하는 메서드; 원래 문자열에서 `separator`를 기준으로 분리하여 최대 `limit`개의 문자열을 반환한다. `separator`가 주어지지 않으면 분리되지 않으며, `limit`이 주어지지 않으면 반환하는 문자열 배열의 길이에 제한이 없다.

[10] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String

- `substring(indexStart[, indexEnd]) => string`: 문자열의 일부분을 추출하는 메서드; `indexStart`번째 문자부터 `indexEnd`번째 문자 직전까지의 문자열을 추출하여 반환하며, `indexEnd`의 기본값은 `length` 속성값이다.
- `toLowerCase() => string`: 문자열 중 알파벳을 모두 소문자로 바꾸어 반환하는 메서드
- `toUpperCase() => string`: 문자열 중 알파벳을 모두 대문자로 바꾸어 반환하는 메서드
- `trim() => string`: 문자열의 앞뒤 whitespace를 모두 제거하여 반환하는 메서드

Code 4.31 String Object Methods

```
> const str = '  Hello World!\t';
> str.length;
15
> str.includes('ll');
true
> str.replace('World', 'Korea');
'  Hello Korea!\t'
> str.split('o');
[ '  Hell', ' W', 'rld!\t' ]
> str.substring(6);
'o World!\t'
> str.toUpperCase();
'  HELLO WORLD!\t'
> str.trim();
'Hello World!'
```

Array Objects

배열에서 자주 쓰이는 속성이나 메서드이며, MDN 문서[11]에서 더 자세하게 확인할 수 있다. 많은 메서드가 high-order 함수임을 확인할 수 있다.

- `length`: 배열의 길이를 값으로 갖는 속성
- `concat([arr1, arr2, ..., arrN]) => Array`: 원래 배열의 뒤에 여러 배열을 이어붙여 반환하는 메서드
- `filter(fn(element[, index, array]) => boolean) => Array`: 배열의 원소 중 주어진 test를 통과한 원소들의 배열을 반환하는 메서드. 배열의 원소 중 `fn`의 반환값이 `true`인 원소들의 배열을 반환한다. `fn`은 원소의 값, index, 원본 배열을 인자로 받아 boolean 값을 반환한다.
- `find(fn(element[, index, array]) => boolean) => Any`: 배열의 원소 중 주어진 함수를 만족하는 첫 번째 원소를 반환하는 메서드
- `findIndex(fn(element[, index, array]) => boolean) => number`: 배열의 원소 중 주어진 함수를 만족하는 첫 번째 원소의 index를 반환하는 메서드
- `forEach(fn(element[, index, array]))`: 배열의 각 원소에 대해 주어진 함수를 실행하는 메서드
- `includes(searchElement[, fromIndex]) => boolean`: 배열의 원소에 주어진 원소가 있는지 판별하는 메서드; 배열의 `fromIndex`번째 원소 이후에 `searchElement`가 있으면 `true`를 반환한다. `fromIndex`의 기본값은 `0`이다.
- `join([separator]) => string`: 배열의 모든 원소를 주어진 문자열로 이어붙인 문자열을 반환하는 메서드

[11] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array

- `map(fn(element[, index, array]) => Any) => Array`: 배열의 각 원소에 대한 주어진 함수의 반환값의 배열을 반환하는 메서드
- `pop() => Any`: 배열의 마지막 원소를 제거하고, 마지막 원소의 값을 반환하는 메서드
- `push([val0, val1, ..., valN]) => number`: 배열의 끝에 주어진 값을 배열의 원소로 차례대로 삽입하고, 값이 모두 삽입된 배열의 길이를 반환하는 메서드
- `reverse() => Array`: 배열의 원소들의 순서를 반대로 뒤집어 반환하는 메서드; 원본 배열도 뒤집힌다는 점을 주의하여야 한다.
- `slice([start, end]) => Array`: 배열의 일부분을 추출하는 메서드; `start`번째 원소부터 `end`번째 원소 직전까지의 원소를 반환하며, `start`의 기본값은 `0`, `end`의 기본값은 `length` 속성이다.
- `sort([compareFn(firstEl, secondEl) => number])`: 배열의 원소들을 주어진 함수에 따라 정렬하여 반환하는 메서드; `compareFn`은 두 비교 대상 원소를 인자로 받으며, 반환값이 음수인 경우 `firstEl`이, 양수인 경우 `secondEl`이 앞에 정렬된다. `compareFn`이 주어지지 않은 경우 알파벳 순서대로 정렬된다. 원본 배열도 정렬된다는 점을 주의하여야 한다.

Code 4.32 Array Methods

```
> const arr = ['apple', 'orange', 'banana'];
> arr.length;
3
> arr.concat(['cherry', 'grapes']);
[ 'apple', 'orange', 'banana', 'cherry', 'grapes' ]
> arr.filter(fruit => fruit.length === 6);
[ 'orange', 'banana' ]
> arr.findIndex(fruit => fruit[0] === 'b');
2
> arr.includes('lemon');
false
> arr.join(' + ');
'apple + orange + banana'
> arr.map((elmt, index) => '#' + index + ': ' + elmt);
[ '#0: apple', '#1: orange', '#2: banana' ]
> arr.pop();
'banana'
> arr.push('cherry', 'grapes');
4
> arr.reverse();
[ 'grapes', 'cherry', 'orange', 'apple' ]
> arr.sort();
[ 'apple', 'cherry', 'grapes', 'orange' ]
> arr.sort((first, second) => {
      const firstLength = first.split('e')[0].length;
      const secondLength = second.split('e')[0].length;
      return firstLength - secondLength;
  });
[ 'cherry', 'apple', 'grapes', 'orange' ]
```

Date Objects

Date 클래스는 날짜와 시간을 다루는 클래스이며, Date 클래스의 인스턴스를 생성하기 위해서는 생성자를 이용하여야 하며, Date 클래스의 생성자는 네 가지가 있다.

- new Date(): 현재 날짜/시각을 나타내는 Date 객체를 생성한다.
- new Date(msec): Epoch 시각[12]으로부터 msec밀리초가 지난 후의 날짜/시각을 나타내는 Date 객체를 생성한다.
- new Date(dateStr): dateStr을 분석하여 문자열이 표현하는 날짜/시각을 나타내는 Date 객체를 생성한다.
- new Date(year, mon[, day, hrs, min, sec, msec]): year년 mon월 day일 hrs시 min분 sec초 msec밀리초에 해당하는 날짜/시각을 나타내는 Date 객체를 생성한다. mon의 값은 0이 1월, 11이 12월이다.

Date 객체의 메서드 몇 가지를 나열하면 다음과 같으며, MDN 문서[13]에서 더 자세하게 확인할 수 있다.

- now() => number: Epoch 시각으로부터 현재까지 흐른 시간을 밀리초 단위로 반환하는 정적 메서드
- getFullYear() => number: Date 객체의 날짜/시각의 연도를 반환하는 메서드
- getMinutes() => number: Date 객체의 날짜/시각의 분(分)을 반환하는 메서드
- getUTCDate() => number: Date 객체의 날짜/시각의 UTC 날짜를 반환하는 메서드
- getUTCHours() => number: Date 객체의 날짜/시각의 UTC 시각을 반환하는 메서드
- toLocaleString() => string: Date 객체의 날짜/시각을 현재 지역에서 나타내는 형태의 문자열로 변환하여 반환하는 메서드

Code 4.33 Date Object Methods

```
> const date = new Date(2021, 6, 19);
undefined
> date;
2021-07-18T15:00:00.000Z
> date.getFullYear();
2021
> date.getHours();
0
> date.getUTCHours();
15
> date.toLocaleString();
'2021. 7. 19. 오전 12:00:00'
> const current = new Date();
undefined
> current.getMonth();
4                                   // result may vary
> current.getMinutes();
52                                  // result may vary
> current.getMilliseconds();
584                                 // result may vary
```

[12] 1970년 1월 1일 자정 UTC
[13] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date

Math Objects

`Math` 객체는 수학 연산과 관련된 객체이며, 모든 속성과 메서드가 정적이라는 특징을 갖는다. `Math` 객체의 속성과 메서드 몇 가지를 소개하며, MDN 문서[14]에서 더 자세하게 확인할 수 있다.

- `E`: 자연상수 $e(\approx 2.718)$를 값으로 갖는 속성
- `LN2`: $\log 2(\approx 0.693)$를 값으로 갖는 속성
- `PI`: 원주율 $\pi(\approx 3.142)$를 값으로 갖는 속성
- `SQRT2`: $\sqrt{2}(\approx 1.414)$를 값으로 갖는 속성
- `abs(x) => number`: 주어진 값의 절댓값($|x|$)을 반환하는 메서드
- `ceil(x) => number`: 주어진 값보다 큰 가장 작은 정수(올림, $\lceil x \rceil$)를 반환하는 메서드
- `cos(x) => number`: 주어진 값의 코사인 값($\cos x$)을 반환하는 메서드 (`x`의 단위는 radian)
- `exp(x) => number`: 주어진 값의 exponential 값(e^x)을 반환하는 메서드
- `floor(x) => number`: 주어진 값보다 작은 가장 큰 정수(버림, $\lfloor x \rfloor$)를 반환하는 메서드
- `log(x) => number`: 주어진 값의 자연로그 값($\log x$)을 반환하는 메서드
- `max([val1, val2, ..., valN]) => number`: 주어진 값들 중 가장 큰 값을 반환하는 메서드
- `min([val1, val2, ..., valN]) => number`: 주어진 값들 중 가장 작은 값을 반환하는 메서드
- `pow(x, y) => number`: x^y의 값을 반환하는 메서드[15]
- `random() => number`: 0 이상 1 미만의 랜덤값을 반환하는 메서드
- `round(x) => number`: 주어진 값에 가장 가까운 정수(반올림, $\lfloor x + 0.5 \rfloor$)를 반환하는 메서드
- `sqrt(x) => number`: 주어진 값의 square root 값($\sqrt{x}$)을 반환하는 메서드

Code 4.34 Math Object Methods

```
> Math.SQRT2;
1.4142135623730951
> Math.max(3, 4, 5);
5
> 0.5 * Math.PI * Math.pow(3, 2);
14.137166941154069
Math.tan(Math.PI / 3) * Math.sqrt(3);
2.9999999999999987
> Math.ceil((Math.random() * 10) + 10);
17                                  // result may vary
```

[14] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math
[15] ES7 이후에는 4.3절에서 다룬 지수 연산자를 사용하는 것이 더 간결하다.

4.6 Basics of JS Exercises

Exercise 1

인자로 받은 값이 1 이상 9 이하의 정수인지 판별하여 결과를 반환하는 함수 `isValidNumber`를 구현하여라. `isValidNumber` 함수는 **Code 4.35**와 같이 동작하여야 한다.

Code 4.35 Exercise 1 Example

```
> isValidNumber(9);
true
> isValidNumber('4');
true
> isValidNumber('abc');
false
> isValidNumber(-5);
false
> isValidNumber(3.5);
false
> isValidNumber(3 / 0);
false
```

Exercise 2

인자로 받은 정수의 모든 양의 약수(約數, divisor) 배열을 작은 순서대로 반환하는 함수 `getDivisors`를 구현하여라. 정수 x의 약수는 $\sqrt{x}$까지만 탐색하여도 모두 구할 수 있음을 이용하고, 배열의 `sort` 메서드를 이용하여라. `getDivisors` 함수는 **Code 4.36**과 같이 동작하여야 하며, 인자로 받은 값이 유효한 값인지 확인할 필요는 없다.

Code 4.36 Exercise 2 Example

```
> getDivisors(5);
[ 1, 5 ]
> getDivisors(24);
[ 1, 2, 3, 4, 6, 8, 12, 24 ]
> getDivisors(196);
[ 1, 2, 4, 7, 14, 28, 49, 98, 196 ]
```

Exercise 3

Code 4.37에 주어진 `ellipse` 객체에 타원의 넓이, 둘레의 길이, 이심률을 구하여 반환하는 함수 `getArea`, `getPerimeter`, `getEccentricity`를 구현하여라. 타원의 `width`를 w, `height`를 h라고 하였을 때($w \geq h$), 각 값을 구하는 식은 다음과 같다.

$$\text{(Area)} = \pi w h$$

$$\text{(Perimeter)} \approx 2\pi\sqrt{\frac{w^2+h^2}{2}}$$

$$\text{(Eccentricity)} = \sqrt{1-\left(\frac{h}{w}\right)^2}$$

Code 4.37 Exercise 3

```
> const ellipse = {
      width: 10,
      height: 5,
  };
> ellipse.getArea();
157.07963267948966
> ellipse.getPerimeter();
49.6729413289805
> ellipse.getEccentricity();
0.8660254037844386
```

Chapter 5

Javascript: Dynamic Frontend

Contents

5.1 Javscript with Front-end

이번 장에서는 4장에서 학습한 JS의 기본 문법을 이용하여 HTML 문서의 동적 기능의 구현을 학습한다.

Application of JS on HTML

Front-end에서 JS는 HTML 문서를 제어하고 수정하기 위해 존재하는 언어이므로 CSS와 마찬가지로 HTML 문서에 연동되어야 한다. JS 코드를 HTML 문서에 대해 실행하는 방법은 두 가지가 있는데, 첫 번째 방법은 HTML 문서에서 `script` 태그를 만들어 내부에 JS 코드를 작성하는 것이다. **Code 5.1**을 웹 브라우저에서 열면 개발자 도구의 Console 탭에서 `Hello World!` 문구가 출력된 것을 확인할 수 있다.

Code 5.1 Internal Application of JS

```
<script>
    console.log('Hello World!');
</script>

<h1>HTML and JS</h1>
```

두 번째 방법은 별도의 JS 파일을 만들어, JS 파일에는 JS 스크립트만 작성하고, HTML 파일에는 웹페이지의 구조만 작성하는 방법이다. HTML 파일에는 **Code 5.2**와 같이 `script` 태그의 `src` 속성에 JS 파일의 주소를 작성한다.

Code 5.2 External Application of JS - HTML

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

<h1>HTML and JS</h1>
```

script.js 파일을 **Code 5.3**과 같이 작성한다.

Code 5.3 External Application of JS - JS

```
console.log('Hello World!');
```

이렇게 HTML과 JS 파일을 작성하고 HTML 파일을 열면 **Code 5.1**과 동일한 결과를 확인할 수 있다. 다만, 웹 브라우저는 HTML 문서의 앞부분부터 로딩하므로 JS 파일이 연동된 `script` 태그 뒤에 작성된 HTML 요소를 JS 코드가 인식하지 못할 수 있다는 점을 주의하여야 한다. 이를 방지하기 위해 `script` 태그를 `head` 태그가 아닌 `body` 태그 내부의 중간이나 끝에 작성하기도 하는데, 이 방식은 HTML의 가독성을 떨어뜨린다.

`script` 태그에 `async`나 `defer` 속성을 활성화하면 이러한 문제를 해결할 수 있다. `async` 속성을 속성값 없이 명시하면 해당 JS 코드는 비동기적으로 실행되고, `defer` 속성을 속성값 없이 명시하면 해당 JS 코드는 문서 로딩이 완전히 끝난 뒤에 실행된다.

5.2 Document Object Model (DOM)

문서 객체 모델(Document Object Model; DOM)은 HTML 문서의 프로그래밍 인터페이스이자 구조를 나타내는 모델이며, DOM을 이용하여 HTML 문서로부터 원하는 HTML 요소들을 선택하고, 선택된 요소들의 정보에 접근하고 수정하는 등 HTML 문서를 제어할 수 있다. 예를 들어 DOM을 이용하여 HTML 요소의 내부 텍스트(inner text), 속성과 속성값 등을 읽거나 수정할 수 있다.

HTML 문서의 구조

HTML은 원소들의 계층 구조로 되어 있어 객체로 나타내기 적합하므로 DOM은 HTML 문서를 객체 형태로 표현한다.

Code 5.4 HTML Tree Structure - HTML

```
<!doctype html>
<html>
<head>
    <title>Document Object Model</title>
    <meta name="viewport" content="width=device-width, initial-scale=1" >
</head>
<body>
    <div class="container" id="title">
        <h1>Document Object Model</h1>
    </div>
    <div class="container" id="content">
        <p style="color: red">This article is about Document Object Model.</p>
        <p>Document Object Model can be abbreviated to DOM.</p>
    </div>
</body>
</html>
```

간단한 형태의 HTML 문서인 **Code 5.4**를 객체의 형태로 나타낼 수 있겠는가?

Code 5.5 HTML Tree Structure - Object

```
const htmlDocument = [
  {
    tag: 'html',
    children: [
      {
        tag: 'head',
        children: [
          {
            tag: 'title',
            text: 'Document Object Model'
          }, {
            tag: 'meta',
            attributes: {
              name: 'viewport',
              content: 'width=device-width, initial-scale=1',
            },
```

```
          },
        ],
      }, {
        tag: 'body',
        children: [
          {
            tag: 'div',
            attributes: { class: ['container'], id: 'title' },
            children: [{ tag: 'h1', text: 'Document Object Model' }],
          }, {
            tag: 'div',
            attributes: { class: ['container'], id: 'content' },
            children: [
              {
                tag: 'p',
                attributes: { style: 'color: red' },
                text: 'This article is about Document Object Model.',
              }, {
                tag: 'p',
                text: 'Document Object Model can be abbreviated to DOM.'
              },
            ],
          },
        ],
      },
    ],
  },
];
```

Code 5.5는 **Code 5.4**를 간단한 객체 형태로 나타낸 예시이다. (절대 DOM의 구조는 아니다.)

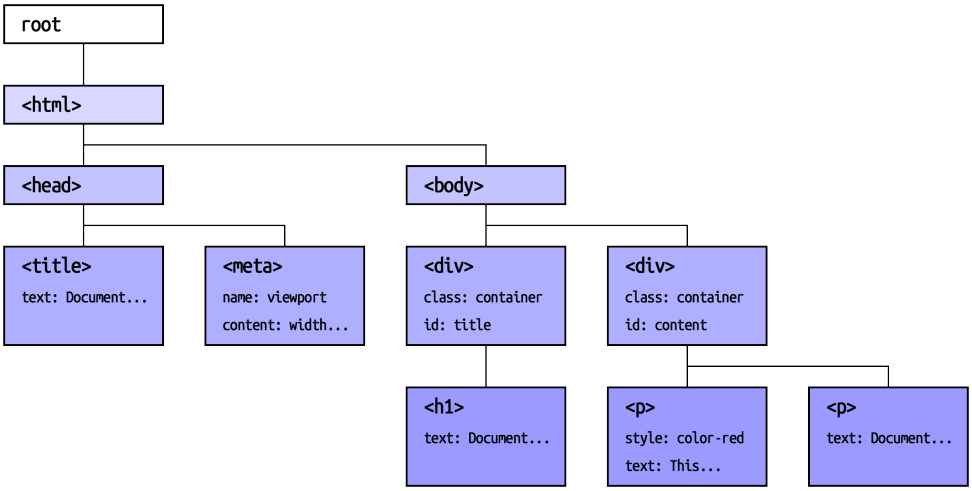


Figure 5.1 Tree Structure of HTML Elements

Code 5.4는 **Figure 5.1**과 같은 구조로 표현될 수도 있다. 즉 HTML 문서는 tree로 표현될 수 있고, 모든 HTML 요소는 tree의 node로 표현될 수 있으며, `document` 객체에 접근하여 tree의 root node에 접근할 수 있다. DOM은 root node를 비롯한 모든 node, 즉 HTML 요소에 대해 어떤 요소의 정보에 접근하는 메서드, 어떤 요소를 기준으로 다른 요소를 선택할 수 있는 메서드 등을 제공한다.

이번 절에서는 DOM에서 제공하는 메서드를 이용하여 HTML 요소들을 제어하는 방법에 대해 학습한다.

HTML 요소 조회

DOM에서 제공하는 메서드를 이용하여 HTML 요소를 조회(query)할 수 있다. **Code 5.6**을 웹 브라우저에서 열고, 개발자 도구의 REPL Shell을 이용하여 DOM에서 제공하는 메서드를 사용해보며 HTML 문서가 어떻게 수정되는지 확인해보자.

Code 5.6 DOM Example HTML

```
<div id="container">
    <h1>Articles List</h1>
    <div class="article-title">
        <a href="/article/5">Ch 5. JS: Dynamic Frontend</a>
    </div>
    <div class="article-title">
        <a class="anchor current" href="/article/4">Ch 4. Basics of Javascript</a>
    </div>
    <div class="article-title">
        <a href="/article/3">Ch 3. CSS: Designing HTML</a>
    </div>
    <div class="article-title">
        <a href="/article/2">Ch 2. HTML: The Basic Structure</a>
    </div>
    <div class="article-title">
        <a href="/article/1">Ch 1. Introduction to Front-end</a>
    </div>
</div>
```

먼저, 어떤 요소의 자손 요소 중 id, class, 태그, CSS 선택자 등을 기준으로 요소를 조회할 수 있다.

- `getElementById(id) => HTMLElement`: 요소의 자손 요소 중 주어진 문자열을 id로 갖는 요소를 반환하는 메서드
- `getElementsByTagName(tagName) => HTMLCollection`: 요소의 자손 요소 중 주어진 문자열을 태그로 갖는 요소들의 유사 배열을 반환하는 메서드
- `getElementsByClassName(className) => HTMLCollection`: 요소의 자손 요소 중 주어진 문자열을 class로 갖는 요소들의 유사 배열을 반환하는 메서드
- `querySelector(selector) => HTMLElement`: 요소의 자손 요소 중 주어진 선택자로 선택되는 첫 번째 요소를 반환하는 메서드
- `querySelectorAll(selector) => NodeList`: 요소의 자손 요소 중 주어진 선택자로 선택되는 요소들의 유사 배열을 반환하는 메서드

유사 배열이란 for문 등으로 iteration을 수행할 수 있으나 배열은 아닌 `HTMLCollection`, `NodeList` 클래스의 인스턴스를 표현하기 위한 용어로, 일반적으로 사용되는 용어는 아니다. 두 클래스 모두 요소들의 개수를 확인할 수 있는 `length` 속성이 있으나, `HTMLCollection` 클래스와 달리 `NodeList` 클래스는 `forEach` 메서드를 갖는다.

Code 5.7을 실행해보며 결과를 확인해보자. 각 표현식의 결과값은 생략한다.

Code 5.7 DOM Query Descendents Examples

```
> document.getElementById('container');
> document.getElementsByClassName('article-title');
> const aTagsList1 = document.getElementsByTagName('a');
> const aTagsList2 = document.querySelectorAll('div.article-title > a');
> aTagsList1;
> aTagsList2;
> for (const aTag of aTagsList1) console.log(aTag);
> aTagsList2.forEach(elmt => console.log(elmt));
> const title = document.getElementsByClassName('article-title')[2];
> title.getElementsByTagName('a');
```

또한, 어떤 요소를 기준으로 상대적인 위치에 있는 요소를 조회할 수 있다.

- `parentElement`: 요소의 부모 요소를 값으로 갖는 속성
- `children`: 요소의 자식 요소들의 유사 배열을 값으로 갖는 속성
- `firstElementChild`: 요소의 첫 번째 자식 요소를 값으로 갖는 속성
- `lastElementChild`: 요소의 마지막 자식 요소를 값으로 갖는 속성
- `previousElementSibling`: 요소의 직전 형제 요소를 값으로 갖는 속성
- `nextElementSibling`: 요소의 직후 형제 요소를 값으로 갖는 속성

Code 5.8을 실행해보며 결과를 확인해보자. 각 표현식의 결과값은 생략한다.

Code 5.8 DOM Query Adjacents Examples

```
> const container = document.getElementById('container');
> const title = container.children[1];
> title.parentElement;
> title.children;
> title.firstElementChild;
> container.lastElementChild;
> title.previousElementSibling;
> title.nextElementSibling;
```

HTML 요소 및 속성 접근 및 수정

요소 객체에 접근하여 속성을 읽거나 쓰고 수정할 수 있다.

- `innerText`: 요소 내부의 텍스트에 접근할 수 있는 속성; 이 속성의 값을 변경하면 요소 내부 텍스트도 변경된다.
- `setAttribute(name, value)`: 요소의 `name` 속성의 값을 `value`로 설정(변경)하는 메서드
- `getAttribute(name) => string`: 요소의 `name` 속성에 대한 속성값을 반환하는 메서드
- `hasAttribute(name) => boolean`: 요소에 `name` 속성이 있는지 판별하는 메서드
- `removeAttribute(name)`: 요소에서 `name` 속성을 제거하는 메서드
- `style`: 요소의 CSS 스타일에 접근할 수 있는 객체; 각 CSS 속성은 `kebab-case`가 아니라 `camelCase`로 작성한다. (**Code 5.9** 참고)

- `classList`: 요소의 클래스의 유사 배열에 접근할 수 있는 객체
  - `length`: 클래스의 개수를 값으로 갖는 속성
  - `value`: 요소의 `class` 속성값을 값으로 갖는 속성
  - `contains(class) => boolean`: 클래스의 존재 유무를 판별하여 반환하는 메서드
  - `add(class1, class2, ..., classN)`: 주어진 클래스들을 추가하는 메서드
  - `remove(class1, class2, ..., classN)`: 주어진 클래스들을 제거하는 메서드
  - `toggle(class) => boolean`: 주어진 클래스가 클래스 유사배열에 존재하면 제거하고, 존재하지 않으면 삽입하며, 주어진 클래스가 추가되었으면 `true`, 제거되었으면 `false`를 반환하는 메서드

Code 5.9 Accessing Attributes of HTML Elements

```
> const title = document.getElementsByClassName('article-title')[1];
> const link = title.getElementsByTagName('a')[0];
> link.innerText = 'JS Grammar';
> link.getAttribute('href');
> link.setAttribute('href', '/article/40');
> link.removeAttribute('href');
> title.hasAttribute('href');
> title.style.color = 'red';
> title.style.borderTop = '1px solid black';
> link.classList;
> link.add('class1', 'class2');
> link.remove('anchor');
> link.toggle('class2');
> link.toggle('class2');
```

또한, 요소에 자식 요소를 추가하거나 제거할 수 있다.

- `createElement(tagName) => element`: 주어진 태그 이름을 태그로 한 새로운 HTML 요소를 생성하여 반환하는 메서드; `document` 객체만 사용할 수 있는 메서드
- `appendChild(element) => element`: 주어진 요소를 요소의 자식 요소로 삽입하고, 삽입된 요소를 반환하는 메서드
- `removeChild(element) => element`: 주어진 요소를 요소의 자식 요소에서 제거하고, 제거된 요소를 반환하는 메서드

Code 5.10 Modifying Child Elements

```
> const container = document.getElementById('container');
> const newButton = document.createElement('button');
> newButton.type = 'submit';
> newButton.innerText = 'Submit';
> container.appendChild(newButton);
> const title = document.getElementsByClassName('article-title')[1];
> container.removeChild(title);
```

5.3 Browser Object Model (BOM)

브라우저 객체 모델(Browser Object Model; BOM)은 웹 브라우저의 정보에 접근하여 읽고 제어할 수 있도록 하는 객체 모델이다. BOM은 DOM과는 달리 표준이 없으나, 현대의 브라우저들은 서로 같은 BOM 속성과 메서드를 가지고 출시되므로 브라우저별로 다른 BOM 코드를 작성할 필요가 없다.

`window` 객체

DOM에서 HTML 문서의 구조와 관련된 모든 객체가 포함된 `document` 객체와 유사하게, BOM에서 브라우저와 관련된 객체들은 모두 `window` 객체에 포함되어 있다. 예를 들어, **Code 4.1**에서 보았던 `alert` 역시 `window` 객체의 메서드이며, `document` 객체 역시 `window` 객체의 하위 객체이다.

Code 5.11 BOM Example

```
> window.alert('This is alert 1');
> alert('This is alert 2');
```

Code 5.11과 같이 `window` 객체의 속성이나 메서드는 `window` 객체를 명시하지 않아도 접근하거나 호출할 수 있다. 이러한 이유로 `document`가 `window`의 하위 객체임에도 불구하고 `window.document`로 작성하지 않고 `document`로 작성하여도 접근할 수 있다.

다음은 `window` 객체에서 유용하게 사용되는 세 가지 메서드이다.

- `alert([message])`: 주어진 문자열 경고 메시지로 하는 대화 상자를 띄우는 메서드
- `confirm([message]) => boolean`: 주어진 문자열을 확인 메시지로 하는 대화 상자를 띄워, [확인]이 눌리면 `true`, [취소]가 눌리면 `false`를 반환하는 메서드; 남용하지 않는 것이 중요하다
- `open(url, windowName[, windowFeature]) => window`: `url`에 위치한 리소스를 `windowName`을 이름으로 하는 새로운 창을 열어 새로운 창 객체를 반환하는 메서드
- `close()`: 현재 창을 닫는 메서드

Code 5.12 `window` Object Methods

```
> confirm('Cancel order?');
> open('https://www.google.com', 'Window Name', 'width=500, height=400');
> close();
```

`screen` 객체

`window.screen` 객체는 현재 웹 브라우저가 위치한 화면의 크기에 관한 객체이다.

- `width`: 화면의 너비를 값으로 갖는 속성
- `height`: 화면의 높이를 값으로 갖는 속성
- `availWidth`: 화면에서 작업 표시줄 등을 제외하고 사용할 수 있는 영역의 너비를 값으로 갖는 속성
- `availHeight`: 화면에서 작업 표시줄 등을 제외하고 사용할 수 있는 영역의 높이를 값으로 갖는 속성
- `colorDepth`: 화면이 나타낼 수 있는 색상의 수를 bit 단위로 나타낸 값을 갖는 속성; 예를 들어 24 bits로 표현 가능한 색상은 $2^{24} \approx 1.68 \times 10^7$ 가지이다.

`location` 객체

`window.location` 객체는 현재 웹 페이지의 주소(위치)에 대해 다루는 객체이며, `hostname`, `pathname`, 프로토콜 등은 back-end 스터디의 HTTP 부분에서 다룰 예정이다.

- `hostname`: 현재 페이지 호스트의 도메인 이름을 값으로 갖는 속성
- `href`: 현재 페이지의 주소를 값으로 갖는 속성
- `pathname`: 현재 페이지의 경로를 값으로 갖는 속성
- `protocol`: 현재 페이지의 프로토콜(HTTP, HTTPS 등)을 값으로 갖는 속성
- `port`: 현재 페이지의 포트를 값으로 갖는 속성
- `assign(url)`: 주어진 문자열을 주소로 하는 웹 문서를 새로 여는 메서드

`history` 객체

`window.history` 객체는 방문한 웹 페이지의 목록을 다루는 객체이다.

- `back()`: 현재 페이지의 이전 페이지를 여는 메서드
- `forward()`: 현재 페이지 다음 페이지를 여는 메서드.
- `go(delta)`: 현재 페이지를 기준으로 주어진 페이지 수만큼 떨어진 페이지를 여는 메서드; `delta=2`라면 두 페이지 앞, `delta=-2`이면 두 페이지 뒤의 페이지를 열며, `delta=0`이거나 주어지지 않은 경우 현재 페이지를 reload한다.

Timeout and Interval

웹 페이지를 작성할 때 공이 포물선 형태로 날아가거나, 일정 시간마다 데이터를 가져와 갱신하는 기능 등 특정한 시간 간격으로 기능이 수행되게끔 구현해야 할 때가 있다. BOM에서는 이러한 로직을 쉽게 생성하고 다룰 수 있는 메서드를 제공한다.

먼저, `setTimeout` 메서드는 함수 `fn`과 시간 `delay`를 인자로 받아, 이 로직을 제어할 수 있는 값 `timeoutId`를 반환하고, 메서드 호출 `delay` 밀리초 후 함수 `fn`을 실행하는 메서드이다. `clearTimeout` 메서드는 `timeoutId`를 인자로 받아 해당 로직을 중단하는 메서드이다.

Code 5.13 Timeout Methods

```
> const timeoutId = setTimeout(() => {
      console.log('setTimeout 3000');
  }, 3000);
> clearTimeout(timeoutId);
```

또한, `setInterval` 메서드는 함수 `fn`과 시간 `delay`를 인자로 받아, 이 로직을 제어할 수 있는 값 `intervalId`를 반환하고, 메서드 호출 `delay` 밀리초 이후부터 함수 `fn`을 `delay` 밀리초마다 반복적으로 실행하는 메서드이다. `clearInterval` 메서드는 `intervalId`를 인자로 받아 해당 로직을 중단하는 메서드이다.

Code 5.14 Interval Methods

```
> const helloLoopId = setInterval(() => {
      console.log('Hello');
  }, 400);
> const worldLoopId = setInterval(() => {
    console.log('World');
  }, 1000);
> clearInterval(helloLoopId);
> clearInterval(worldLoopId);
```

5.4 Event and Event Listener

프로그램에서 **이벤트(event)**란 개념은 어떠한 사건을 의미하며, front-end에서 발생할 수 있는 이벤트에는 요소를 클릭하는 행위, 키보드를 이용하여 키를 입력하는 행위, 드래그하는 행위 등이 있다. 이러한 사건이 일어나면 "이벤트가 발생했다"라고 표현한다.

이벤트를 이용하면 버튼을 눌러 원하는 함수를 작동시키고, 키보드를 이용하여 게임 오브젝트를 컨트롤 하는 등 사용자에 의해 동적으로 작동하는 웹 페이지를 구현할 수 있다. 이러한 기능은 특정 이벤트가 발생하였을 때 실행되어야 하는 작업을 명시하여 구현하며, 이렇게 특정 요소에서 특정 이벤트가 발생하였을 때 실행될 함수를 **이벤트 리스너(event listener)**라고 한다.

이번 절에서는 이벤트와 이벤트 리스너를 이용하여 웹 페이지를 구현하는 방법을 다룬다.

Events

웹 페이지에서 발생할 수 있는 이벤트의 종류는 다양하며, 본 교재에서는 자주 쓰이는 몇 가지만 소개한다. 먼저 마우스 동작과 관련된 이벤트이다.

- `click`: 요소가 좌클릭되었을 때 발생하는 이벤트
- `mousedown`: 요소 위에 커서를 대고 마우스를 눌렀을 때 발생하는 이벤트
- `mouseenter`: 마우스 커서가 요소 안으로 들어올 때 발생하는 이벤트
- `mouseleave`: 마우스 커서가 요소 밖으로 나갈 때 발생하는 이벤트

다음은 키보드를 이용한 키 입력과 관련된 이벤트이다.

- `keydown`: 키보드의 키를 눌렀을 때 발생하는 이벤트
- `keyup`: 키보드의 키를 떼었을 때 발생하는 이벤트
- `keypress`: 키보드의 키를 눌렀을 때 발생하는 이벤트; Ctrl, Alt 등의 기능키에는 작동하지 않음

소개된 이벤트 외의 이벤트의 종류는 매우 많으므로 필요에 따라 찾아서 사용하는 것이 좋다. 웹 페이지에서 발생할 수 있는 이벤트는 아래의 링크에서 확인해볼 수 있다.

- https://www.w3schools.com/jsref/dom_obj_event.asp

Event Listener

특정 요소에서 이벤트가 발생했을 때 이벤트 리스너를 실행시키기 위해서는 해당 요소와 이벤트 리스너를 등록해야 하며, 그 방법은 크게 세 가지가 있다.

먼저, HTML 요소의 속성과 속성값으로 직접 명시하는 방법이다. 먼저 속성의 이름은 이벤트의 이름 앞에 접두사 `on`을 붙인다. 예를 들어 `click` 이벤트에 대응하는 속성은 `onclick`, `keypress` 이벤트에 대응하는 속성은 `onkeypress`이다. 이후 속성값에 이벤트가 발생했을 때 수행될 JS 코드를 작성한다. 이러한 방식으로 이벤트 리스너를 연동하면 특정 요소에서 특정 이벤트가 발생하였을 때 어떤 작업이 수행되는지 직관적으로 파악할

수 있으나, 3.2절에서 다룬 CSS의 inline style의 단점과 마찬가지로 HTML 문서의 목적에 위배되며, 유지 및 보수의 관점에서 매우 비효율적이다.

Code 5.15 Event Listener - Inline Method

```
<button id="btn" onclick="console.log('Clicked!');">Click Button</button>
```

두 번째 방법은 HTML 요소 객체에 이벤트 리스너를 메서드로 할당하는 방법이다. 메서드의 이름은 앞의 방식과 마찬가지로 접두사 `on`을 붙여 작성하고, 메서드의 값에 이벤트 리스너를 할당한다. 이 방식은 JS 파일에 작성할 수 있어 HTML의 목적에 위배되지 않고, IE 8 이전 버전의 브라우저에서도 호환되며, 간결하다는 장점을 갖는다. 그러나 IE 8[1] 이전의 브라우저는 지나치게 오래되어 지원할 필요성이 너무 적고, 한 요소의 한 이벤트에 하나의 이벤트 리스너만 등록할 수 있다는 단점이 있다.

Code 5.16 Event Listener – Overriding Method

```
document.getElementById('btn').onclick = () => {
    console.log('Clicked!');
};
```

마지막 방법은 요소의 `addEventListener` 메서드를 사용하는 방법으로, `addEventListener` 메서드는 이벤트의 이름과 이벤트 리스너를 인자로 받아, 이벤트 리스너를 추가하는 메서드이다. 이렇게 `addEventListener` 메서드를 이용한 방법이 가장 권장되는 방법이다.

Code 5.17 Event Listener – Using `addEventListener`

```
document.getElementById('btn').addEventListener('click', () => {
    console.log('Clicked!');
});
```

Event Object

지금까지 이벤트의 종류와 이벤트 리스너를 등록하는 방법에 대해 다루었다. 그런데 지금까지 학습한 방법으로는 이벤트 리스너에서 발생한 이벤트에 대한 정보를 얻을 수 없다는 문제가 발생한다. 예를 들어 `keypress` 이벤트가 발생했을 때 사용자가 어느 키를 눌렀는지, `mousedown` 이벤트가 발생했을 때 사용자가 좌클릭했는지 우클릭했는지, 어느 위치에 커서를 대고 눌렀는지 등에 대한 정보를 얻지 못한다.

이러한 문제는 이벤트 리스너가 이벤트 객체를 받음으로써 해결된다. 발생한 이벤트에 대한 여러 정보를 담은 이벤트 객체가 생성되어 이벤트 리스너에 전달되며, 이 이벤트 객체를 인자로 받아서 사용할 수 있다. JS의 함수의 특성상 이벤트 리스너에서 이벤트 객체가 필요하지 않다면 앞의 **Code 5.17**과 같이 인자를 아예 받지 않을 수도 있다.

[1] 2009년 3월에 발표되었다

Code 5.18 Event Listener with Event Object

```
document.addEventListener('mousedown', event => {
    console.log(event);
});
```

이벤트 객체의 속성 중 자주 쓰이는 속성을 몇 가지 소개한다.

- `target`: 이벤트가 발생한 요소 객체를 값으로 갖는 속성
- `button`: 마우스 이벤트를 발생시킨 마우스의 버튼 번호를 값으로 갖는 속성; 0, 1, 2, 3, 4는 각각 왼쪽 버튼, 휠 버튼, 오른쪽 버튼, 뒤로 가기 버튼, 앞으로 가기 버튼을 뜻한다.
- `clientX`, `clientY`: 뷰포트를 기준으로 마우스 이벤트가 발생한 위치를 값으로 갖는 속성
- `offsetX`, `offsetY`: 이벤트가 발생한 요소를 기준으로 마우스 이벤트가 발생한 위치를 값으로 갖는 속성
- `screenX`, `screenY`: 웹 페이지가 존재하는 화면을 기준으로 마우스 이벤트가 발생한 위치를 값으로 갖는 속성
- `code`: 키 입력과 관련된 이벤트를 발생시킨 키를 문자열[2]로 나타낸 속성

Code 5.19는 이벤트 객체를 이용하여 `#panel` 요소를 좌클릭 또는 우클릭하였을 때 `#panel` 요소의 위쪽 테두리와 왼쪽 테두리에 내린 수선을 표시되며, 좌클릭하였을 때는 빨간색, 우클릭하였을 때는 파란색 수선이 표시되는 예제이다. (편의상 CSS와 JS를 모두 HTML 문서에 작성하였다.)

Code 5.19 Event Object Example

```
<div id="panel">
    <div id="offset"></div>
</div>

<style>
    #panel {
        width: 800px;
        height: 400px;
        border: 3px solid black;
    }
</style>

<script>
    document.getElementById('panel').addEventListener('mousedown', event => {
        if (event.button !== 0 && event.button !== 2) return;
        const fooElmt = document.getElementById('offset');
        const borderColor = event.button === 0 ? 'red' : 'blue'
        fooElmt.style.borderRight = '1px solid ' + borderColor;
        fooElmt.style.borderBottom = '1px solid ' + borderColor;
        fooElmt.style.width = event.offsetX + 'px';
        fooElmt.style.height = event.offsetY + 'px';
    });
</script>
```

[2]https://keycode.info/에서 확인할 수 있다.

5.5 JS Exercises

Exercise 1

Code 5.20을 참고하여 버튼을 클릭하면 `#color-box`의 배경색을 랜덤하게 바꾸는 함수를 script.5.1.js의 `setRandomBgColor` 함수에 작성하여라. 색상은 R, G, B의 값으로 표현될 수 있고, 각 값은 0 이상 255 이하의 정수임을 이용하여라.

Code 5.20 Exercise 1

```
<!doctype html>
<head>
    <title>JS Exercise 1</title>
    <meta charset="utf-8">
    <style>
        #color-box {
            width: 800px;
            height: 450px;
            border: 2px solid black
        }
    </style>
    <script src="./script.5.1.js" type="text/javascript" defer></script>
</head>
<body>
    <div id="color-box"></div>
    <button onclick="setRandomBgColor()">Change Color</button>
</body>
```

Exercise 2

Code 5.21을 참고하여, 버튼을 클릭하면 `input`에 입력된 값이 1 이상 9 이하의 자연수면 입력된 값의 구구단을 `#times-result`에 출력하고, 아닌 경우에는 `Input Error!`를 출력하는 함수를 script.5.2.js의 `displayTimes` 함수에 작성하여라. `input` 태그에 입력된 값은 `value` 속성을 통해 접근할 수 있음을 이용하여라.

Code 5.21 Exercise 2

```
<!doctype html>
<head>
    <title>JS Exercise 2</title>
    <meta charset="utf-8">
    <script src="./script.5.2.js" type="text/javascript" defer></script>
</head>
<body>
    <h1>Times Table</h1>
    <h3>Input an integer between 1 - 9.</h3>
    <div>
        <input type="text" id="number" name="number">
        <button onclick="displayTimes()">Show Result</button>
    </div>
    <div id="times-result"></div>
</body>
```

Exercise 3

Code 5.22를 참고하여 버튼을 클릭하면 다섯 개 상자 중 두 번째 상자의 색을 변경하는 함수를 script.5.3.js의 `changeBoxColor` 함수에 작성하여라.

Code 5.22 Exercise 3

```
<!doctype html>
<head>
    <title>JS Exercise 3</title>
    <meta charset="utf-8">
    <style>
        .box {
            display: inline-block;
            width: 100px;
            height: 100px;
            background-color: red
        }
    </style>
    <script src="./script.5.3.js" type="text/javascript" defer></script>
</head>
<body>
    <div>
        <div class="box"></div>
        <div class="box"></div>
        <div class="box"></div>
        <div class="box"></div>
        <div class="box"></div>
    </div>
    <button onclick="changeBoxColor()">Change Color</button>
</body>
```

Exercise 4

Code 5.23을 참고하여 버튼을 눌렀을 때 주어진 레벨업 시스템이 작동하여 정해진 확률에 따라 레벨 0에서 레벨 10까지 50ms마다 레벨업을 계속 시도하는 함수를 script.5.4.js의 `work` 함수에 작성하여라.

- 레벨 0에서 레벨 9까지 다음 레벨로의 레벨업이 성공할 확률은 각각 100%, 60%, 36%, 22%, 13%, 8%, 5%, 3%, 2%, 1%이며, 성공하면 레벨이 1만큼 오르고, 실패하면 변하지 않는다.
- 웹 페이지에는 현재 레벨과 시도 횟수를 표시하며, 현재 레벨이 파란색 게이지 바의 길이에 반영되어야 한다. (레벨 n인 경우 전체 게이지 바의 $\frac{n}{10}$을 차지)

다음 단계에 따라 문제를 해결하여라.

- Step 1: 레벨업 성공 확률을 % 단위, number형의 배열로 저장한다.
- Step 2: 레벨, 시도 횟수 등 필요한 값을 변수와 상수로 선언한다.
- Step 3: % 단위의 확률을 인자로 받아, 확률에 따라 무작위로 성공 여부를 판별하여 반환하는 함수 `getRandomBinaryResult`를 구현한다.
  - 예: `getRandomBinaryResult(32)`의 반환값은 32%의 확률로 `true`, 68%의 확률로 `false`이다.
  - Hint: 0 이상 20 미만의 정수 중 하나를 무작위로 뽑았을 때 13 미만의 정수는 65% 확률로 뽑힌다.
- Step 4: 버튼을 눌렀을 때 작동하는 기능은 `tryLevelUp` 함수 내부에 작성한다.

Code 5.23 Exercise 4

```
<!doctype html>
<head>
    <title>JS Exercise 4</title>
    <meta charset="utf-8">
    <style>
        #gauge-background {
            position: relative;
            width: 300px;
            height: 10px;
            margin: 20px 0;
            background-color: red
        }
        #gauge-bar {
            position: absolute;
            height: 10px;
            background-color: blue
        }
    </style>
    <script src="./script.js" type="text/javascript" defer></script>
</head>
<body>
    <h1>Level-Up System</h1>
    <div id="gauge-background">
        <div id="gauge-bar"></div>
    </div>
    <div>Current Level: <span id="current-level">0</span></div>
    <div>Attempts: <span id="attempts">0</span></div>
    <button id="start-btn" onclick="tryLevelUp()">Start Level Up</button>
</body>
```

Exercise 5

Code 5.24를 참고하여 물품을 카트에 담을 때마다 지불할 총 금액이 갱신되는 쇼핑몰 사이트를 script.5.5.js를 다음 단계를 따라 작성하여 완성하여라.

- Step 1: `.item` 요소들은 그 물품의 이름을 `id` 값으로 가지고 있다. 각 `id`가 `key`, 해당 물품의 단가를 `value`로 하는 객체를 만든다. 그리고, 필요한 변수/상수들을 선언하고 할당한다.
- Step 2: 이 문제에서는 여러 요소에 이벤트 리스너를 등록해야 하므로 등록할 요소들을 선택하고, for-of 문을 사용하여 각 요소에 이벤트 리스너를 등록한다.
- Step 3: 이벤트가 발생한 요소가 속한 `.item` 요소의 `id` 값을 가져와 총 금액을 갱신하는 이벤트 리스너를 구현한다.

Code 5.24 Exercise 5

```
<!doctype html>
<head>
    <title>JS Exercise 5</title>
    <meta charset="utf-8">
    <style>
        body > div { margin: 20px 0 }
        .item {
            width: 500px; padding: 10px; margin: 20px 0;
            border: 2px solid gray; background-color: #d3d3d3;
        }
        .item > div { display: inline-block; vertical-align: top }
        .item-image > img { width: 100px }
        .item-info { margin-left: 10px }
        .item-info > div { margin-bottom: 10px }
        .item-name { font-size: 20px; font-weight: bold }
    </style>
    <script src="./script.5.5.js" type="text/javascript" defer></script>
</head>
<body>
    <div id="header">
        <h1>Marketplace</h1>
        <div>Total Cost: <span id="cost">0</span> KRW</div>
    </div>
    <div id="items">
        <div class="item" id="apple">
            <div class="item-image">
                <img src="https://i.imgur.com/PMmhG47.jpg" >
            </div>
            <div class="item-info">
                <div class="item-name">Apple</div>
                <div class="item-price">Unit Price: 700 KRW</div>
                <button class="add-to-cart">Add to Cart</button>
            </div>
        </div>
        <div class="item" id="orange">
            <div class="item-image">
                <img src="https://i.imgur.com/IOEmEO1.jpg" >
            </div>
            <div class="item-info">
                <div class="item-name">Orange</div>
                <div class="item-price">Unit Price: 800 KRW</div>
                <button class="add-to-cart">Add to Cart</button>
            </div>
        </div>
        <div class="item" id="lemon">
            <div class="item-image">
                <img src="https://i.imgur.com/ZYLR99S.jpg" >
            </div>
            <div class="item-info">
                <div class="item-name">Lemon</div>
                <div class="item-price">Unit Price: 900 KRW</div>
                <button class="add-to-cart">Add to Cart</button>
            </div>
        </div>
    </div>
</body>
```

Exercise 6

Code 5.25와 **Code 5.26**을 참고하여 미로를 탈출 게임을 script.5.6.js를 작성하여 완성하여라. 시작 지점은 붉은 사각형, 탈출 지점은 푸른 사각형, 미로의 벽은 검정색 사각형이며, 붉은 사각형을 방향키를 이용하여 탈출 지점까지 이동시킨다. 방향키를 이용해 이동할 때는 인접한 사각형으로 이동하며, 벽이나 미로 밖으로 이동하려고 하면 이동하지 않는다.

- Step 0: 이 문제의 미로는 `#maze` 요소 내부에 정사각형 요소가 7행 8열로 배치되어 있다. 사각형의 위치는 행과 열을 이용하여 나타내고, `row`와 `col`을 속성으로 갖는 위치 객체를 이용한다. 예를 들어 탈출 지점은 1행 7열에 위치하며, 탈출 지점의 위치 객체는 `{ row: 1, col: 7 }`이다.
- Step 1: 행과 열의 최솟값과 최댓값을 나타내는 상수를 선언하고, 붉은 사각형의 위치 객체를 선언한다.
- Step 2: 위치 객체를 인자로 받아 해당 위치에 있는 요소를 반환하는 함수 `getElementByPosition`을 구현한다.
- Step 3: 이벤트 객체의 키 입력값을 인자로 받아 붉은 사각형의 위치에서 키 입력에 따라 이동한 새로운 위치 객체를 반환하는 함수 `getNewPositionByKey`를 구현한다.
- Step 4: 인자로 받은 위치 객체가 미로 내에 존재하는지 판별하여 반환하는 함수 `isPositionInRange`를 구현한다.
- Step 5: 인자로 받은 위치 객체가 미로의 벽인지 판별하여 반환하는 함수 `isPositionWall`을 구현한다.
- Step 6: 방향키를 떼었을 때의 이벤트 리스너를 구현한다. 먼저 현재 위치의 사각형을 붉은 사각형에서 해제하고, 키 입력값을 이용하여 새로운 위치를 구한다. 새로운 위치가 적절한 위치이면 현재 위치를 새로운 위치로 바꾸고, 새로운 위치의 사각형을 붉은 사각형으로 지정한다. 마지막으로 새로운 위치가 탈출 지점이면 `You Escaped!` 문구를 경고창으로 띄운다.

Code 5.25 Exercise 6 - CSS

```
#maze {
    margin: 30px;
    font-size: 0
}

.cell {
    display: inline-block;
    width: 50px;
    height: 50px
}

.wall {
    background-color: black
}

.target {
    background-color: deepskyblue
}

.current {
    background-color: lightcoral
}
```

Code 5.26 Exercise 6 - HTML

```
<html>
<head>
    <title>JS Exercise 6</title>
    <meta charset="utf-8">
    <link rel="stylesheet" text="type/css" href="./style.5.6.css">
    <script src="./script.5.6.js" type="text/javascript" defer></script>
</head>
<body>
    <h1>Escape Simple Maze!</h1>
    <div id="maze">
        <div class="cells">
            <div class="cell wall"></div><div class="cell wall"></div>
            <div class="cell wall"></div><div class="cell wall"></div>
            <div class="cell wall"></div><div class="cell wall"></div>
            <div class="cell wall"></div><div class="cell wall"></div>
        </div>
        <div class="cells">
            <div class="cell wall"></div><div class="cell"></div>
            <div class="cell"></div><div class="cell wall"></div>
            <div class="cell"></div><div class="cell"></div>
            <div class="cell"></div><div class="cell target"></div>
        </div>
        <div class="cells">
            <div class="cell wall"></div><div class="cell"></div>
            <div class="cell wall"></div><div class="cell wall"></div>
            <div class="cell"></div><div class="cell wall"></div>
            <div class="cell wall"></div><div class="cell wall"></div>
        </div>
        <div class="cells">
            <div class="cell wall"></div><div class="cell"></div>
            <div class="cell"></div><div class="cell"></div>
            <div class="cell"></div><div class="cell"></div>
            <div class="cell"></div><div class="cell wall"></div>
        </div>
        <div class="cells">
            <div class="cell wall"></div><div class="cell wall"></div>
            <div class="cell wall"></div><div class="cell wall"></div>
            <div class="cell wall"></div><div class="cell wall"></div>
            <div class="cell"></div><div class="cell wall"></div>
        </div>
        <div class="cells">
            <div class="cell current"></div><div class="cell"></div>
            <div class="cell"></div><div class="cell"></div>
            <div class="cell"></div><div class="cell"></div>
            <div class="cell"></div><div class="cell wall"></div>
        </div>
        <div class="cells">
            <div class="cell wall"></div><div class="cell wall"></div>
            <div class="cell wall"></div><div class="cell wall"></div>
            <div class="cell wall"></div><div class="cell wall"></div>
            <div class="cell wall"></div><div class="cell wall"></div>
        </div>
    </div>
</body>
</html>
```

Appendix A

Exercise Answers

Contents

A.1 HTML Exercise Answers

Exercise 1

Code A.1 Exercise 1 Answer

```
<!doctype html>
<html>
<head>
    <title>KWEB Survey Page</title>
</head>
<body>
    <div id="survey-header">
        <h2>KWEB 설문조사</h2>
        <div id="kweb-logo">
            <img src="http://info.korea.ac.kr/_res/info/img/community/img_kweb.gif"
                 alt="KWEB Logo" width=250>
        </div>
        <a href="https://www.facebook.com/kwebfamily/">KWEB 페이스북 페이지</a>
    </div>
    <div id="survey-body">
        <div class="survey-question">
            <h3>1. 이름을 작성해주세요.</h3>
            <input type="text" name="name">
        </div>
        <div class="survey-question">
            <h3>2. 학번을 작성해주세요.</h3>
            <input type="text" name="studentNumber">
        </div>
        <div class="survey-question">
            <h3>3. 현재 회원 등급을 선택해주세요.</h3>
            <input type="radio" name="level" value="intern">준회원
            <input type="radio" name="level" value="qualified">정회원
            <input type="radio" name="level" value="break">휴회원
        </div>
        <div class="survey-question">
            <h3>4. 본인이 관심있는 분야를 모두 선택해주세요.</h3>
            <input type="checkbox" name="field" value="frontend">프론트엔드 프로그래밍
            <input type="checkbox" name="field" value="backend">백엔드 프로그래밍
            <input type="checkbox" name="field" value="crawling">웹 크롤링
            <input type="checkbox" name="field" value="security">웹 보안
            <input type="checkbox" name="field" value="etc">기타
            <input type="text" name="fieldText">
        </div>
        <div class="survey-question">
            <h3>
                5. 2020년 봄학기
                KWEB 스터디에 대한 전반적인 평가와 피드백을 작성해주세요.
            </h3>
            <textarea name="feedbacks" cols="100" rows="20"></textarea>
        </div>
    </div>
    <div id="survey-footer">
        <button type="submit">제출</button>
    </div>
</body>
</html>
```

A.2 CSS Exercise Answers

Exercise 1

Code A.2 Exercise 1 Answer

```
h1 {
    text-align: center;
}

.chapter-name {
    font-weight: bold
}

#current-chapter > .chapter-name {
    color: red
}

.chapter > .horizontal-lists {
    color: blue
}
```

Exercise 2

Code A.3 Exercise 2 Answer

```
.numbers {
    margin: 20px 0;
    text-align: center;
}

.number {
    display: inline-block;
    width: 50px;
    padding-top: 12px;
    padding-bottom: 15px;
    margin: 0 10px;
    border: 1px solid black;
    border-radius: 25px;
    background-color: lightcoral;
}
```

Exercise 3

Code A.4 Exercise 3 Answer (HTML)

```
<div id="scroll-button">
    <a href="#page-top">
        <div id="top-scroll-cell" class="scroll-button-cell">&#9650;</div>
    </a>
    <a href="#page-bottom">
        <div class="scroll-button-cell">&#9660;</div>
    </a>
</div>
```

Code A.5 Exercise 3 Answer (CSS)

```
#scroll-button {
    position: fixed;
    right: 30px;
    bottom: 30px;
    border: 3px solid darkblue;
    border-radius: 18px;
    background-color: darkblue;
    font-size: 15pt;
}

.scroll-button-cell {
    padding: 1px 7px 4px;
    color: white;
}

.scroll-button-cell:not(#top-scroll-cell) {
    border-top: 2px solid white;
}
```

Exercise 4

Code A.6 Exercise 4 Answer

```
h1 {
    color: white;
}

body {
    background-color: skyblue;
}

@media only screen and (min-width: 600px) {
    body {
        background-color: blue;
    }
}

@media only screen and (min-width: 900px) {
    h1 {
        color: yellow;
    }
}

@media only screen and (min-width: 1200px) {
    body {
        background-color: darkblue;
    }
}
```

Exercise 5

Code A.7 Exercise 5 Answer

```
div {
    margin: 20px 0;
}

.header {
    height: 100px;
    background-color: pink;
    border: 2px solid red;
}

.navbar {
    height: 200px;
    background-color: gold;
    border: 2px solid brown;
}

.content {
    height: 300px;
    background-color: skyblue;
    border: 2px solid blue;
}

.footer {
    height: 200px;
    background-color: lightgreen;
    border: 2px solid green;
}

@media only screen and (min-width: 800px) {
    .body > div {
        margin: 0;
    }

    .navbar {
        width: 19%;
        height: 500px;
        display: inline-block;
    }

    .content {
        width: 79%;
        height: 500px;
        display: inline-block;
    }
}
```

A.3 Basics of JS Exercise Answers

Exercise 1

Code A.8 Exercise 1 Answer

```
const isValidNumber = num => {
    const parsedNumber = parseInt(num);
    if (!isFinite(parsedNumber) || isNaN(parsedNumber)) return false;
    if (parsedNumber != num) return false;
    if (parsedNumber < 1 || parsedNumber > 9) return false;
    return true;
}
```

Exercise 2

Code A.9 Exercise 2 Answer

```
const getDivisors = num => {
    const divisors = [];
    for (let i = 1; i <= Math.sqrt(num); i++) {
        if (i * i === num) divisors.push(i);
        else if (num % i === 0) divisors.push(i, num / i);
    }
    return divisors.sort((first, second) => first - second);
}
```

Exercise 3

Code A.10 Exercise 3 Answer

```
const ellipse = {
    width: 10,
    height: 5,
    getArea() {
        return Math.PI * this.width * this.height;
    },
    getPerimeter() {
        return 2 * Math.PI * Math.sqrt((this.height ** 2 + this.width ** 2) / 2);
    },
    getEccentricity() {
        return Math.sqrt(1 - (this.height / this.width) ** 2);
    },
};
```

A.4 JS Exercise Answers

Exercise 1

Code A.11 Exercise 1 Answer

```
const setRandomBgColor = () => {
    const red = Math.floor(Math.random() * 256);
    const blue = Math.floor(Math.random() * 256);
    const green = Math.floor(Math.random() * 256);
    const newColor = 'rgb(' + red + ',' + blue + ',' + green + ')';
    document.getElementById('color-box').style.backgroundColor = newColor;
};
```

Exercise 2

Code A.12 Exercise 2 Answer

```
const displayTimes = () => {
    const number = document.getElementById('number').value;
    const parsedNumber = parseInt(number);
    const timesResult = document.getElementById('times-result');

    if (!isFinite(parsedNumber) || isNaN(parsedNumber)) {
        return timesResult.innerText = 'Input Error!';
    }

    if (parsedNumber != number) {
        return timesResult.innerText = 'Input Error!';
    }

    if (parsedNumber < 1 || parsedNumber > 9) {
        return timesResult.innerText = 'Input Error!';
    }

    const result = [];
    for (let i = 1; i <= 9; i++) {
        result.push(parsedNumber + ' X ' + i + ' = ' + parsedNumber * i);
    }
    timesResult.innerText = result.join('\n');
};
```

Exercise 3

Code A.13 Exercise 3 Answer

```
const changeBoxColor = () => {
    const box = document.getElementsByClassName('box')[1];
    box.style.backgroundColor = 'blue';
};
```

Exercise 4

Code A.14 Exercise 4 Answer

```
const probs = [ 100, 60, 36, 22, 13, 8, 5, 3, 2, 1 ];

let attempts = 0;
let level = 0;

const getRandomBinaryResult = ratio => {
    const randomNumber = Math.floor(Math.random() * 100);
    return (randomNumber < ratio);
};

const tryLevelUp = () => {
    const loopId = setInterval(() => {
        attempts++;
        const successed = getRandomBinaryResult(probs[level]);
        if (successed) {
            level++;
            document.getElementById('gauge-bar').style.width = level * 10 + '%';
            document.getElementById('current-level').innerText = level;
        }
        document.getElementById('attempts').innerText = attempts;
        if (level >= 10) clearInterval(loopId);
    }, 50);
};
```

Exercise 5

Code A.15 Exercise 5 Answer

```
const itemPrice = {
    apple: 700,
    orange: 800,
    lemon: 900,
};
let totalPrice = 0;

for (const buttonEl of document.getElementsByClassName('add-to-cart')) {
    buttonEl.addEventListener('click', event => {
        const itemName = event.target.parentNode.parentNode.getAttribute('id');
        totalPrice += itemPrice[itemName];
        document.getElementById('cost').innerText = totalPrice;
    });
}
```

Exercise 6

Code A.16 Exercise 6

```
const ROW_MIN = 0;
const ROW_MAX = 6;
const COL_MIN = 0;
const COL_MAX = 7;
const currentPos = { row: 5, col: 0 };

const getElementByPosition = pos => {
    const cellsEl = document.getElementsByClassName('cells')[pos.row];
    return cellsEl.getElementsByClassName('cell')[pos.col];
};

const getNewPositionByKey = key => {
    const pos = {
        row: currentPos.row,
        col: currentPos.col,
    }
    switch (key) {
        case 'ArrowUp': pos.row--; break;
        case 'ArrowDown': pos.row++; break;
        case 'ArrowLeft': pos.col--; break;
        case 'ArrowRight': pos.col++; break;
    }
    return pos;
};

const isPositionInRange = pos => (pos.row >= ROW_MIN)
    && (pos.row <= ROW_MAX) && (pos.col >= COL_MIN) && (pos.col <= COL_MAX);

const isPositionWall = pos => getElementByPosition(pos).classList.contains('wall');

document.addEventListener('keyup', event => {
    getElementByPosition(currentPos).classList.remove('current');
    const newPos = getNewPositionByKey(event.code);
    if (isPositionInRange(newPos) && !isPositionWall(newPos)) {
        currentPos.row = newPos.row;
        currentPos.col = newPos.col;
    }
    const newElmt = getElementByPosition(currentPos);
    newElmt.classList.add('current');
    if (newElmt.classList.contains('target')) alert('You escaped!');
});
```