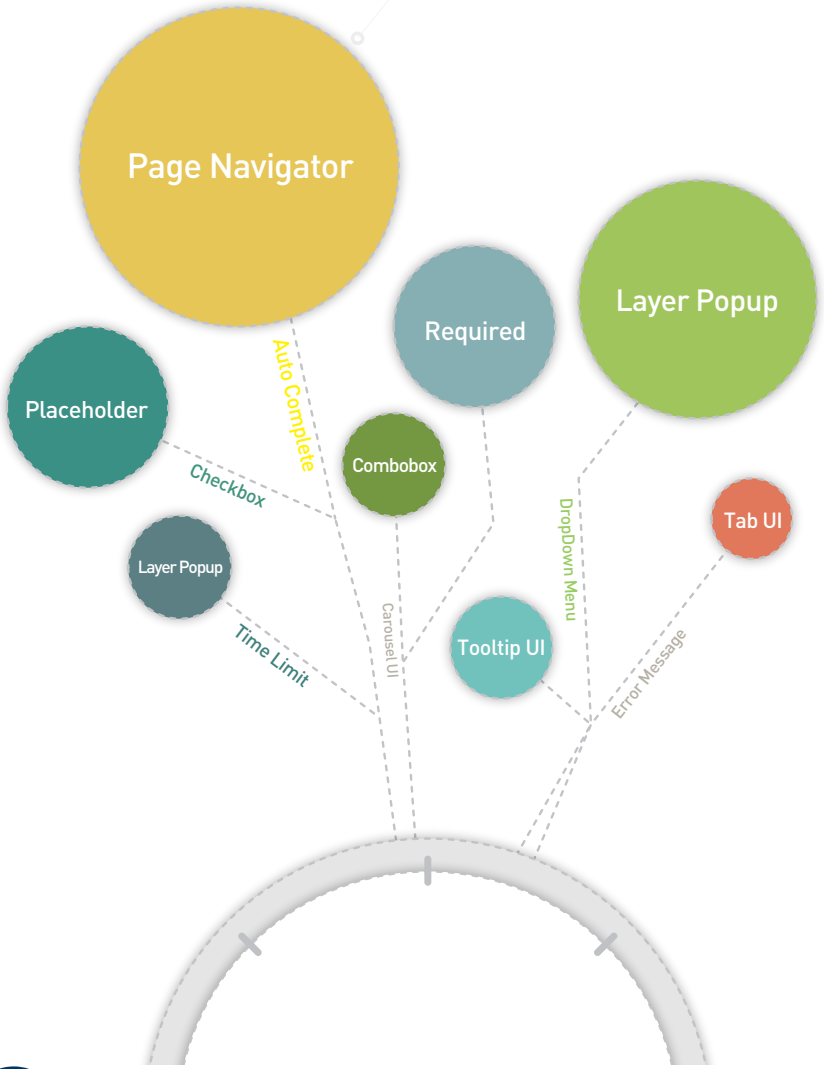
예제로 살펴보는

WAI-ARIA

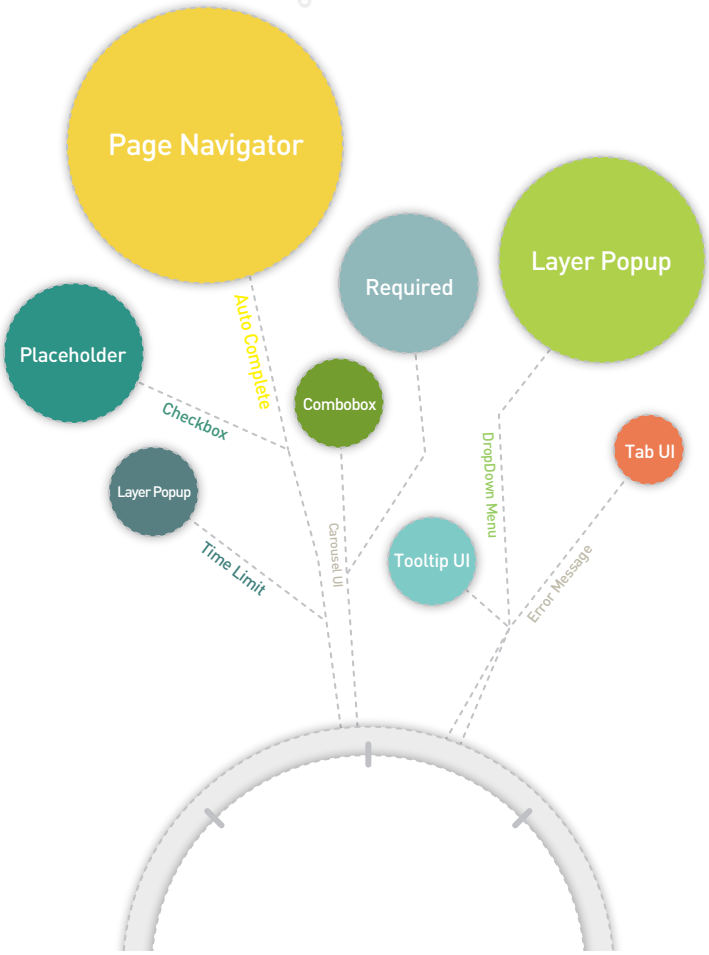
정보접근성 향상을 위한 W3C 국제표준 WAI-ARIA 사례집



미래창조과학부 NIA 한국정보화진흥원

예제로 살펴보는

WAI-ARIA



미래창조과학부 NIA 한국정보화진흥원


Carousel UI
DropDown Menu
Auto Complete UI
Page Navigator
Combobox
Combobox
Tab UI
Placeholder
Layer Popup

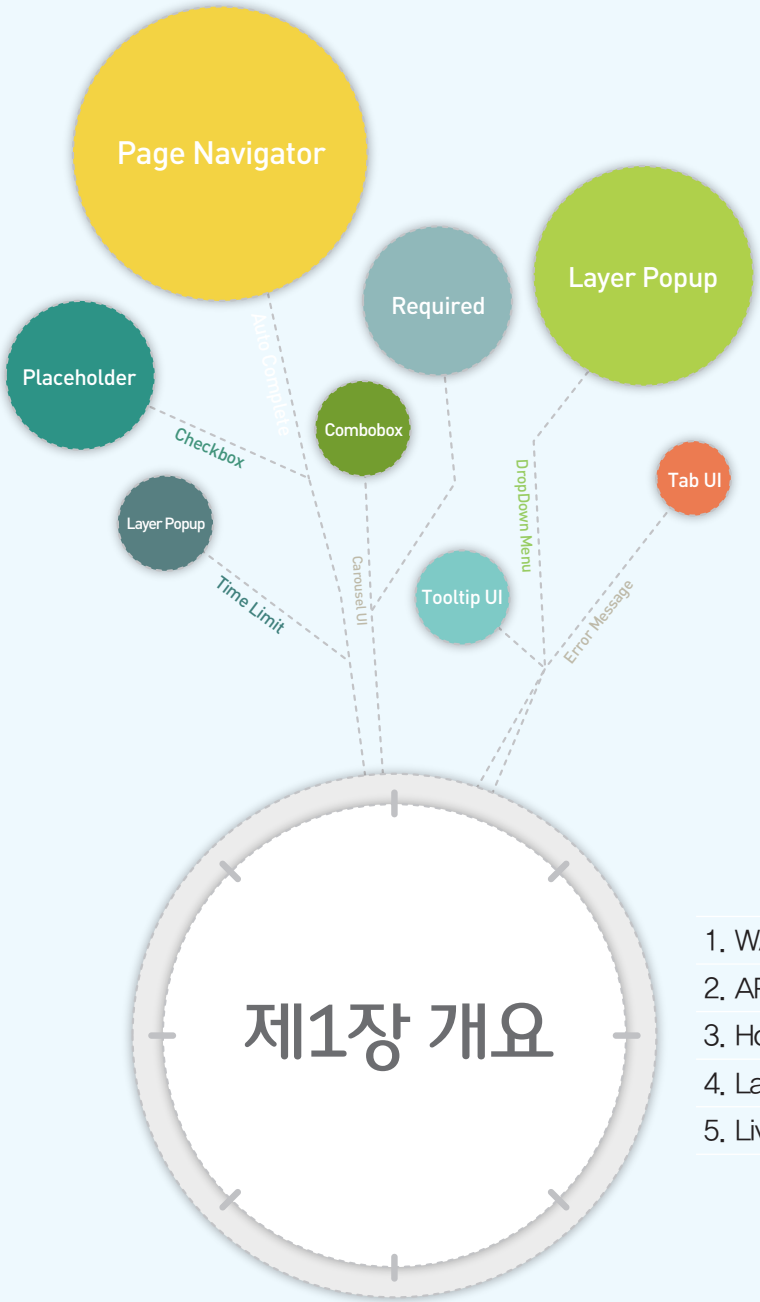
예제로 살펴보는

WAI-ARIA

CONTENTS

제1장 개요

제2장 사례집



제1장 개요

1. WAI-ARIA 소개

1-1. WAI-ARIA 란?

웹은 초기에 문서와 문서를 연결하기 위한 목적으로 설계되었다. 웹의 가장 중요한 개념은 문서간의 연결이었고 이를 하이퍼텍스트(Hyper Text)라는 개념으로 불렀다. 이러한 웹의 등장은 인류의 역사를 크게 바꾸어 놓았으며 물리적인 공간과 시간의 제약을 벗어나 전 세계가 하나로 연결될 수 있게 되었다.

웹의 인기가 날로 높아가고 매우 빠른 속도로 성장하면서 웹이 더 이상 문서의 개념이 아닌 데스크탑 수준의 애플리케이션과 같은 사용자 경험(User eXperience, 이하 UX)을 요구하기에 이르렀다. 이런 요구를 충족시키기 위해 등장한 것이 바로 리치 인터넷 애플리케이션(Rich Internet Applications, 이하 RIA)이다.

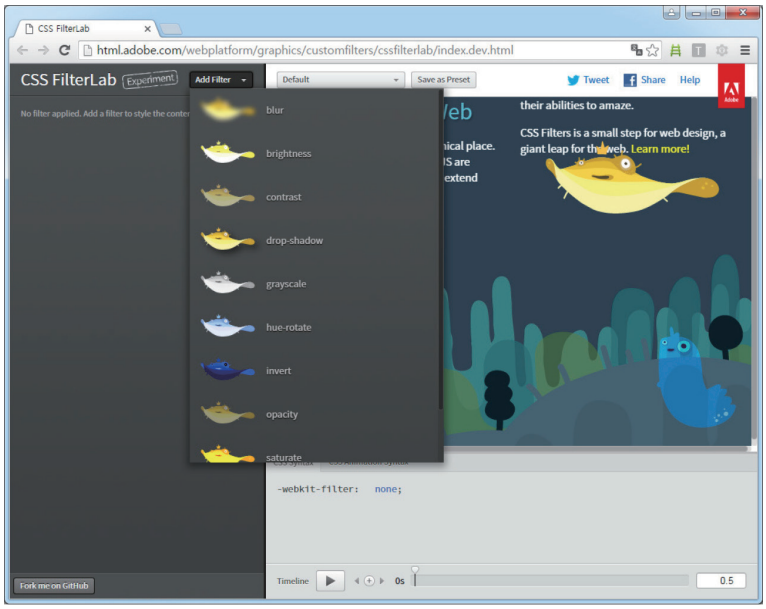
RIA는 기존의 정적인 HTML과 단순한 자바스크립트 사용 환경에서 벗어나 한층 강력해진 자바스크립트와 Ajax(Asyncronous Javascript and XML, 이하 Ajax) 등의 기술을 활용하여 웹 애플리케이션을 제작하고 좀 더 향상된 UX를 제공할 수 있다.

RIA의 등장은 역동적이고 화려하면서 편리함까지 갖춘 UX를 제공하게 되었지만 모든 사용자가 동등하게 접근하고 사용할 수 없는 문제가 있었다. 바로 스크린리더 등 보조기술을 사용하는 장애인이 RIA 기술로 제작된 웹 애플리케이션을 제대로 사용할 수 없었기 때문이다.

예를 들어 자바스크립트와 Ajax 등을 활용하여 웹 애플리케이션을 제작할 때 〈div〉나 〈span〉 등의 의미를 가지지 않는 요소로 특정 컴포넌트를 구현하는 경우이다. 이런 경우 스크린리더 등의 보조기기에서는 해당 컴포넌트의 기능을 명확하게 파악하기 어렵다.

또한 주식 시세나 RSS Feed 등 시간에 따라 정보가 자동으로 업데이트 되는 경우 역시 스크린리더 등 보조기기가 이를 알 수 없어 RIA 기술이 접근성에 취약하다는 비판을 받아왔다.

자바스크립트로 구현된 이미지 필터



이에 W3C(World Wide Web Consortium, 이하 W3C)에서는 웹 콘텐츠 및 웹 애플리케이션의 접근성과 상호 운용성을 개선하기 위해 기술 명세를 발표했는데 이 명세가 바로 WAI-ARIA(Web Accessibility Initiative – Web Accessible Rich Internet Applications, 이하 WAI-ARIA) 이다.

WAI-ARIA는 스크린리더 및 보조기기 등에서 접근성 및 상호 운용성을 향상시키기 위한 목적으로 탄생했으며 웹 애플리케이션에 역할(Role), 속성(Property), 상태(State) 정보를 추가하여 이를 개선 할 수 있도록 제공하고 있다.

2014년 3월 20일 WAI-ARIA는 W3C 권고안으로 지정되었으며, WAI-ARIA 명세의 일부는 HTML5 명세와 통합되었다.

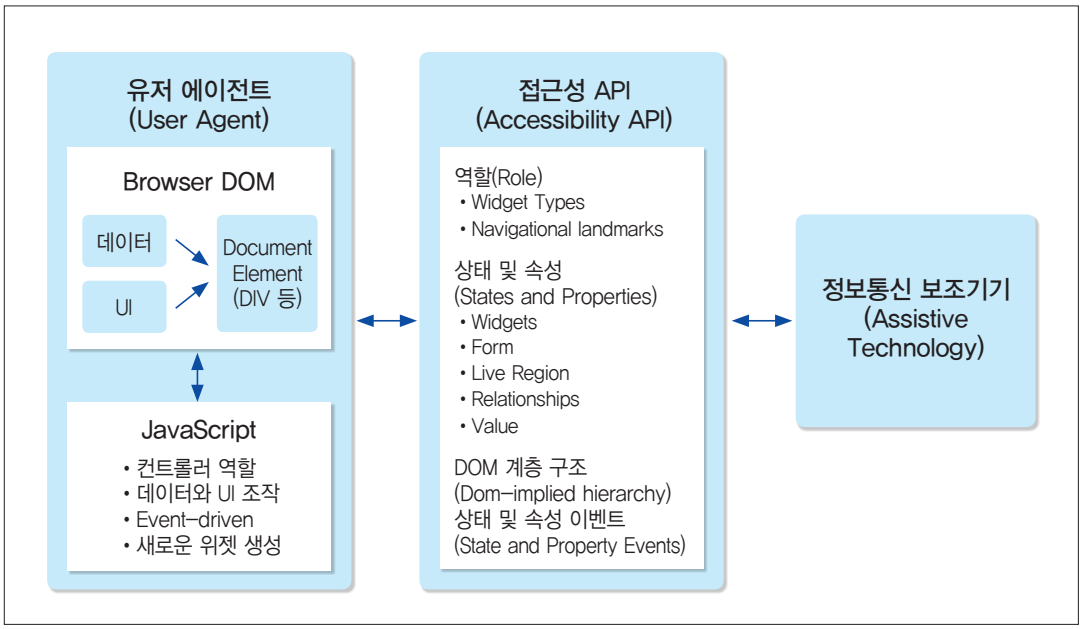
http://www.w3.org/TR/2014/REC-wai-aria-20140320/ (W3C 권고안1.0 – 2014.03.20)

1-2. WAI-ARIA의 목적

WAI-ARIA는 스크린리더 및 보조기기 등에서 접근성 및 상호 운용성을 향상시키기 위해 마크업에 역할(Role), 속성(Property), 상태(State) 정보를 추가할 수 있도록 지원한다. 이렇게 추가된 정보는 웹 브라우저에 의해 자동으로 해석되어 각각의 운영체제(OS)의 접근성 API로 변환되도록 설계되었다.

이때 스크린리더 및 보조기기는 운영체제(OS)에서 제공하는 접근성 API를 통해 데스크탑 애플리케이션과 동일한 방법으로 자바스크립트 컨트롤을 인식하고 상호 작용을 하게 된다. 이것은 스크린리더 및 보조기기 사용자가 웹 애플리케이션을 사용할 때, 데스크탑 애플리케이션의 동작과 유사하게 인식하고 작동하기 때문에 보다 향상된 UX를 제공하게 된다.

유저 에이전트(웹 브라우저)와 접근성 API 및 보조 기술 간의 관계



최근 버전의 웹 브라우저는 WAI-ARIA의 다양한 기능을 잘 지원하고 있다. 그러나 웹 브라우저와 스크린리더의 조합에 따라 WAI-ARIA 지원이 균일하지 못한 상황이다. 가령 방향키 등을 이용하여 항목을 선택하는 경우 NVDA와 JAWS에서는 정상적으로 키보드 인터랙션을 읽어주지만 국산 스크린리더의 대표라 할 수 있는 센스리더는 이를 읽지 못하는 문제가 있다.

또한 aria-live 속성은 JAWS의 버그로 인해 페이지 로드 시 적용되어 동적으로 노드가 삽입되지 않을 경우 정상적으로 읽히지 않는다.

다양한 스크린리더

WAI-ARIA는 개발자의 의도가 보조 기술에 잘 전달될 수 있도록 요소(Element)나 컴포넌트에 누락된 의미 구조를 제공하는 것을 목적으로 한다.

이것을 통해 논리적 구조 설계가 가능해지고 페이지 영역의 빠른 탐색을 제공 할 수 있게 된다. 또한 동적으로 변경되는 콘텐츠의 식별이 가능해지고 상태 변화가 발생했을 때 웹 브라우저와 같은 유저 에이전트는 접근성 API를 활용하여 적절한 이벤트 알림 등을 제공하여 보조기기의 사용성을 향상시킬 수 있다.

2. ARIA Roles & States and Properties

접근성과 상호 운용성을 향상시키기 위한 WAI-ARIA는 역할(Role), 속성(Property), 상태(State) 등의 3가지 기능을 제공한다.

- 역할(Role) – 유저 인터페이스(User Interface, 이하 UI)에 포함된 특정 컴포넌트의 역할을 정의
- 속성(Property) – 해당 컴포넌트의 특징이나 상황을 정의하며 속성명으로 "aria-*"라는 접두사를 사용
- 상태(State) – 해당 컴포넌트의 상태 정보를 정의

2-1. WAI-ARIA Role

먼저 역할(Role)에 대해 살펴보자. 역할(Role)은 특정 요소(Element)에 기능을 정의하는 것을 말한다. 페이지의 검색 영역(Search)인지 내비게이션(Navigation) 요소인지 특정 섹션의 제목(Heading) 인지 등의 명확한 기능을 부여할 수 있다.

예를 들면 웹 애플리케이션에서 버튼 컴포넌트를 제작할 때 〈a〉 요소를 사용하는 경우를 가정해 보자. 〈a〉 요소는 마우스 사용 이외에 키보드 사용 시 초점을 받을 수 있기 때문에 다양한 용도로 활용되고 있다.

그러나 버튼 컴포넌트 구현을 위해 〈a〉 요소를 사용할 경우 스크린리더에서는 해당 버튼을 링크라고 읽어주게 된다. 이럴 경우 스크린리더 사용자는 버튼이 아닌 링크의 용도로 이해하기 때문에 혼란이 생기게 된다.

이때 〈a〉 요소에 role="button"을 지정하면 스크린리더 사용자는 링크가 아닌 버튼으로 읽어주게 되어 컴포넌트의 정확한 용도를 이해하고 사용할 수 있게 된다.

또한 기본 버튼의 키보드 인터랙션으로 사용하기 위해 스페이스바로도 동작을 추가하면 버튼의 역할이 완성된다.

```
<a href="/" onclick="playApp()" role="button">재생</a>
```

WAI-ARIA Role은 다음 4가지의 카테고리로 구분할 수 있다. .

- Document Structure Role (문서구조 역할)
- Abstract Role (추상 역할)
- Landmark Role (랜드마크 역할)
- Widget Role (위젯 역할)

2-2. WAI-ARIA Property & State

속성(Property)은 요소(Element)가 기본적으로 갖고 있는 특징이나 상황을 의미한다. 폼의 입력 상자가 읽기 전용(Read Only) 인지 또는 필수 항목(Require)인지, 사용자 입력에 대해 자동 완성(Auto Complete)기능을 지원할 지 또는 드래그(Drag)가 가능한지, 팝업(has Popup)이 뜨는지, 업데이트된(Live) 정보가 있는지 등의 상황을 사용자가 인지할 수 있도록 할 수 있다.

속성(Property)의 사용 예로 입력 폼을 생각해 보자. 회원 가입 시 사용자 아이디와 비밀번호를 입력 받아야 하며 해당 입력 값은 필수 항목일 경우 입력 상자에 aria-required="true"를 지정하면 스크린리더 등의 보조기기에서 해당 항목이 필수 항목(Require)임을 알 수 있도록 제공할 수 있다.

```
<div class="id-area">
    <label for="user-email">아이디</label>
    <input type="email" id="user-email" aria-required="true">
</div>
<div class="pw-area">
    <label for="user-pw">비밀번호</label>
    <input type="password" id="user-pw" aria-required="true">
</div>
```

마지막으로 상태(State)는 요소(Element)의 현재 상태를 의미하며 상황의 변화에 따른 값을 가진다. 메뉴가 펼쳐진 상태(expanded)인지, 적절하지 못한(invalid) 값이 입력되었는지 콘텐츠가 숨김(hidden) 상태인지 등을 나타낼 수 있다.

상태(State) 관련 간단한 예를 살펴보자. 애플리케이션에서 제공되는 메뉴가 하위 메뉴를 포함하고 있을 경우 현재 하위 메뉴가 접힌 상태인지 펼쳐진 상태인지 스크린리더 사용자에게 정보를 제공 해야 할 경우가 있다. 이때 aria-expanded 속성을 사용하여 접힌 상태라면 false 값을 펼쳐진 상태라면 true 값을 지정할 수 있다.

```
<ul id="menu" role="tree">
    <li id="menu" role="treeitem" aria-expanded="true">
        <a>WAI-ARIA 소개</a>
        <ul id="sub-menu" role="group">
            <li id="menu" role="treeitem" aria-expanded="false">
                <a>WAI-ARIA 란?</a>
            </li>
        </ul>
    </li>
    <li id="menu" role="treeitem" aria-expanded="false">
        <a>WAI-ARIA의 목적</a>
    </li>
</ul>
```

지금까지 WAI-ARIA의 3가지 기능에 대해 살펴보았다. 이렇게 WAI-ARIA의 다양한 기능을 사용하면 웹 애플리케이션의 접근성과 사용성을 개선할 수 있다. 그러나 콘텐츠에 따라 의미에 맞는 요소(Element)를 사용하는 대신 WAI-ARIA에서 제공하는 기능만으로 접근성 문제를 해결하고자 하는 것은 바람직하지 않다.

3. How to Use (WAI-ARIA 작성 규칙)

3-1. 랜드마크와 HTML5

HTML5에 새롭게 추가된 섹션 관련 요소의 경우 WAI-ARIA 규칙과 동일한 역할의 요소가 다수 있다. W3C 에서는 HTML5의 섹션 관련 요소와 WAI-ARIA 규칙을 함께 사용할 경우 해당 기능이 무효화 되거나 충돌이 발생할 수 있으므로 중복해서 사용하지 않도록 주의를 당부하고 있다.

ARIA Role과 HTML5 섹션 관련 요소 비교

Landmark Role	HTML5 섹션 관련 요소
role="application"	동일한 역할의 요소 없음. 주로 〈div〉 요소와 같이 그룹 역할을 하는 요소로 대체할 수 있다.
role="banner"	동일한 역할의 요소 없음. 비슷한 의미로 〈header〉 요소를 사용할 수 있으나 〈header role="banner"〉로 사용하였다면 한 페이지에서 한 개의 〈header〉 요소만 사용하길 권장한다.
role="navigation"	〈nav〉 요소. 다른 페이지 또는 페이지 내 특정 영역으로 이동하는 링크 콘텐츠 영역으로 주로 메인 메뉴 및 서브 메뉴 등에 사용할 수 있다.
role="main"	〈main〉 요소. 본문의 주요 콘텐츠 영역으로 한 페이지 내에 1개만 사용이 가능하며, 〈article〉, 〈aside〉, 〈footer〉 요소의 하위 요소로 사용할 수 없다
role="complementary"	〈aside〉 요소. 주요 콘텐츠와 연관이 적은 의미있는 콘텐츠 영역으로 종종 사이드바로 표현할 수 있다. 〈aside〉 영역에는 현재 날씨, 관련된 기사 또는 주식 정보등의 부가 콘텐츠를 포함 할 수 있다.
role="form"	〈form〉 요소. 폼과 관련된 요소의 모임을 표현하는 영역으로 서버에 전송될 수 있는 콘텐츠를 포함 할 수 있다.
role="search"	동일한 역할의 요소 없음. 검색의 역할을 담당하는 서식 영역임을 의미하며 〈div〉 또는 〈form〉 요소를 사용하는 것을 권장한다.
role="contentinfo"	동일한 역할의 요소 없음. 비슷한 의미로 〈footer〉 요소를 사용할 수 있으나 〈footer role="contentinfo"〉로 사용하였다면 한 페이지에서 한 개의 〈footer〉 요소만 사용하길 권장한다.

3-2. HTML요소의 기능 변경 제한

ARIA 규칙을 이용하여 요소의 네이티브(Native) 의미를 변경하는 것은 바람직하지 않다. 예를 들어 버튼의 역할을 하는 콘텐츠를 〈h1〉 요소로 마크업하고 role="button"으로 지정하는 경우를 들 수 있다.

이미 제공되고 있는 〈h1〉 요소에 버튼을 역할을 수행하도록 하고자 한다면 〈h1〉 요소에 직접 role을 부여하기 보다 자식 요소로 의미에 맞는 〈button〉 요소를 추가하거나 중립적인 의미를 가지는 〈span〉 등의 요소에 role="button"을 부여하여 추가하는 것을 권장한다.

`<h1 role="button"> 버튼 </h1>`	X
`<h1><button> 버튼 </button></h1>`	O
`<h1><span role="button" tabindex="0">버튼</span></h1>`	O

3-3. 키보드 사용 보장

사용자와 상호작용이 필요한 대화형 UI의 경우 키보드로도 접근 및 사용이 가능하도록 제공하여야 한다. 여기서 상호작용이 필요한 대화형 UI란 사용자가 클릭할 수 있는 정보나 탭 또는 드래그 앤 드롭, 슬라이드, 스크롤 등의 기능이 필요한 콘텐츠를 의미한다.

기본적으로 키보드 포커스를 받지 못하는 HTML 요소의 경우 tabindex 속성을 사용하여 키보드 포커스를 받을 수 있도록 할 수 있다. 이때 tabindex 속성에 0을 지정하면 콘텐츠의 선형화 순서대로 키보드 포커스가 진입하게 되고 0보다 작은 값을 지정하면 키보드 포커스를 받을 수 없는 상태가 된다.

`<span role="button" tabindex="-1">버튼</span>`	키보드 포커스 X
`<span role="button" tabindex="0">버튼</span>`	키보드 포커스 O

3-4. 숨김 콘텐츠

사용자에게 정보를 전달하되 단순히 화면에서만 보이지 않도록 처리된 콘텐츠에 aria-hidden="true"를 지정해서는 안 된다. 단순히 가시적으로만 숨긴 콘텐츠에 특정 role을 부여했더라도 스크린리더 등의 보조기기에서는 aria-hidden="true"로 지정된 상태라면 의미적으로도 숨겨진 콘텐츠로 인식하게 된다.

또한 이와 비슷하게 특정한 의미를 전달해야 하는 요소에 presentation 역할(Role)을 주어서는 안된다. 스크린리더 등의 보조기기는 role="presentation"으로 지정된 요소를 의미 없이 단순히 가시적으로 전달하기 위한 요소로 인식하게 된다.

`<button role="presentation"> 버튼 </button>`	X
`<button aria-hidden="true"> 버튼 </button>`	X

aria-hidden="true"를 사용하여 숨김 콘텐츠에 대한 사용자의 접근을 차단하고자 할 경우 CSS의 display 속성에 none 값을 지정하여 스크린리더 등의 보조기기에서 접근할 수 없도록 하고 aria-hidden="true"을 명시해야 한다.

`button { display : none }` `<button aria-hidden="true"> 버튼 </button>`	O

3-5. 레이블 제공

모든 대화형 UI의 경우 반드시 레이블을 제공하여야 한다. 레이블 제공을 위해 HTML의 〈label〉 요소를 사용하는 것을 권장하며 aria-label, aria-labelledby 등의 WAI-ARIA 관련 속성을 사용하여 레이블을 제공할 수도 있다.

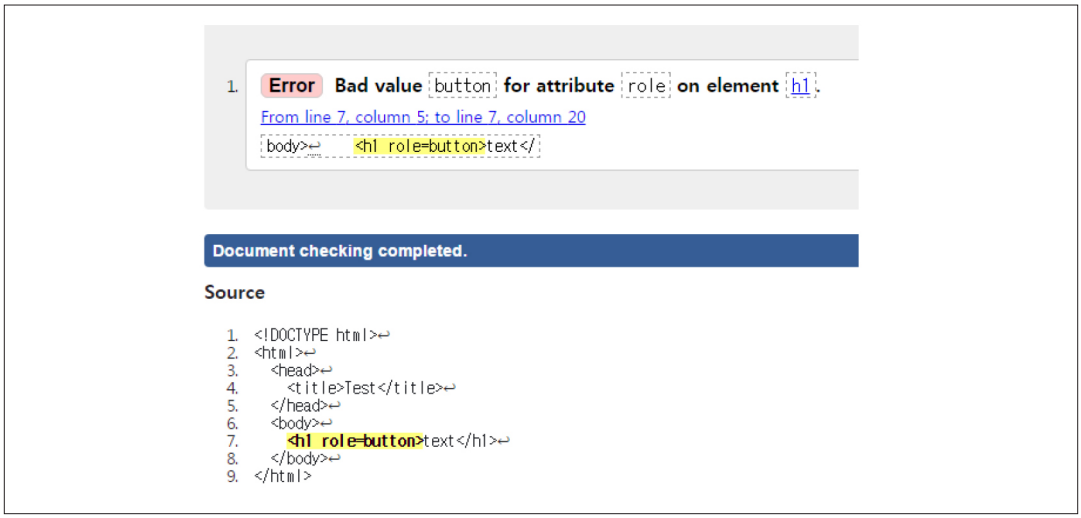
```
<div class="container">
   <label for="user-name">이름</label>
   <input type="text" id="user-name">
</div>

<div>
    <div id="user-name">이름</div>
    <input type="text" aria-labelledby="user-name">
</div>
<button aria-label="닫기" onclick="myDialog.close()"> X </button>
```

3-6. 유효성 검사

WAI-ARIA를 사용할 경우 의미를 가지는 시맨틱 요소와 충돌되지 않도록 하는 것이 필요하다. 가령 헤딩의 의미를 가지는 〈h1〉 요소에 role="button"을 부여하면 해당 요소에 적절하지 않은 속성을 사용했다는 문법 오류를 경험하게 될 것이다.

WAI-ARIA 속성의 사용이 적절하지 않아 문법 오류가 발생한 경우



4. Landmark Role

4-1. 랜드마크(Landmark Role)에 대해

랜드마크(Landmark Role)는 웹 페이지에서 제공되는 콘텐츠 유형이 어떤 역할을 하는지 식별할 수 있도록 도와주는 표지판 기능으로 기존 웹접근성 지침에서 요구하고 있는 건너뛰기 링크의 발전된 모습이면서 콘텐츠 블록의 제목 제공보다 명확한 영역 구분이 가능하다는 장점을 가진다.

랜드마크는 JAWS, NVDA, ORCA, Chromevox, Window Eyes 그리고 Voice Over와 Talkback 및 키보드 사용자를 위한 FireFox Add-on에서 지원하고 있다.

콘텐츠에 랜드마크를 할당하게 되면 스크린리더 사용자는 웹 페이지 영역의 의미와 구조를 명확하게 이해할 수 있고 랜드마크를 탐색하는 핫키를 이용하여 문서의 주요 영역을 자유롭게 탐색할 수 있게 된다.

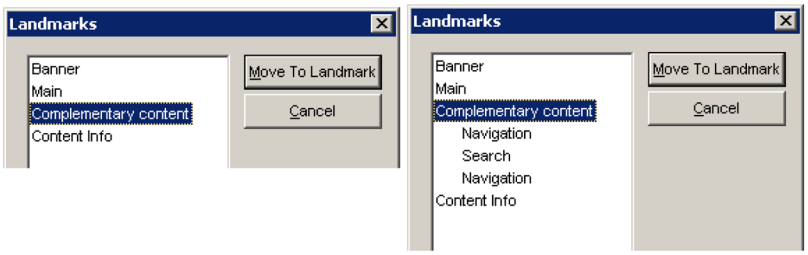
주요 스크린리더의 랜드마크를 탐색하는 핫키는 다음과 같다.

구분	NVDA	JAWS10+	센스리더 4+
다음 (Next Landmark)	D	; (세미콜론)	J
이전 (Previous Landmark)	Shift + D	Shift + ; (세미콜론)	Shift + J

스크린리더 사용자가 핫키를 사용하여 랜드마크를 탐색할 때 “랜드마크 역할 이름” + “landmark” 형식으로 알려주게 되는데 이 경우 아래 방향 화살표를 이용하여 콘텐츠 영역으로 이동할 수 있다.

만약 랜드마크가 다른 랜드마크를 포함하고 있는 컨테이너의 역할이라면 세미콜론을 사용하여 탐색하는 순서에는 포함되지 않지만 리스트 영역에 포함하게 된다. 이때 해당 컨테이너가 포커스를 받게 되면 서브 아이템이 닫혀있는 상태임을 알려주게 되고 사용자는 서브 아이템을 열기 위해 오른쪽 방향 화살표 키를 사용하여 탐색 할 수 있다.

랜드마크가 다른 랜드마크를 포함하고 있는 경우 - JAWS 예시

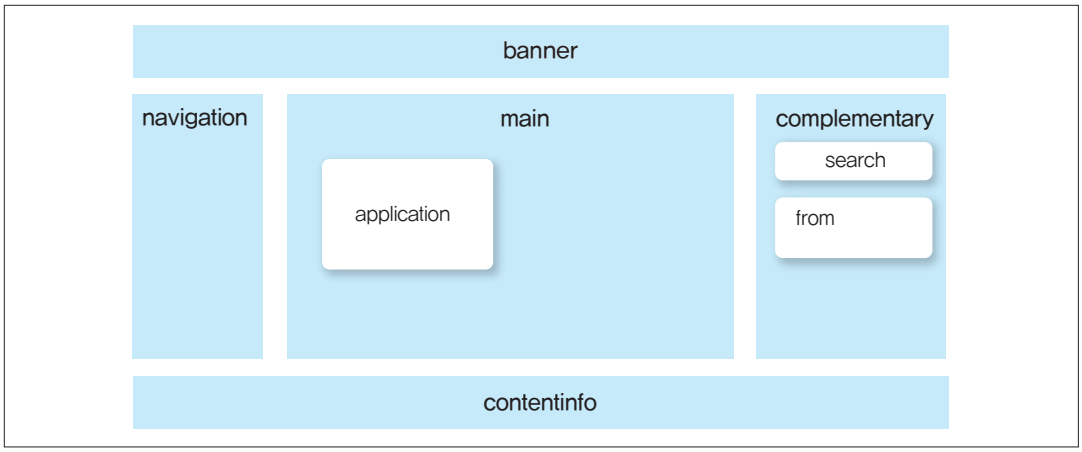


4-2. 랜드마크(Landmark Role) 사용 방법

랜드마크를 추가하는 것은 생각보다 어렵지 않다. 콘텐츠를 포함하고 있는 컨테이너인 HTML 요소에 role 속성을 사용하여 콘텐츠의 역할을 지정하면 된다

```
<div class="container">
   <div role="banner">banner</div>
   <div role="navigation">navigation</div>
   <div role="main">
      <div role="application">application</div>
   </div>
   <div class="complementary">
      <div role="search">search</div>
      <div role="form">form</div>
   </div>
   <div role="contentinfo">contentinfo</div>
</div>
```

랜드마크 사용 예시



4-3. 랜드마크(Landmark Role)의 종류

랜드마크(Landmark Role)는 application, banner, navigation, main, complementary, form, search, contentinfo 등 다음의 8가지 기능을 제공하고 있다.

• application

웹 애플리케이션 영역임을 선언한다. 여기서 말하는 웹 애플리케이션이란 정적인 웹 콘텐츠와 반대되는 개념으로 특정 기능을 제공하는 경우를 의미한다. 스크린리더 등 보조 기기는 role="application"으로 지정된 요소를 만나면 웹 브라우저 탐색 키를 웹 애플리케이션에게 돌려주어야 한다.

• banner

사이트의 로고나 제목 등을 포함하는 헤더 정보를 포함할 수 있는 영역으로 HTML5의 〈header〉 요소와 비슷한 역할을 한다. 이 영역의 콘텐츠는 대체로 사이트 전체에 걸쳐 동일하게 제공된다. 간혹 banner라는 role의 명칭 때문에 광고 배너 등을 삽입하는 영역으로 오해하기도 하는데 banner role의 역할은 주로 브랜딩이나 사이트의 아이덴티티를 나타낼 수 있는 정보를 의미한다는 것을 잊지 말자.

• navigation

웹 사이트의 내비게이션 영역으로 링크 모음을 포함할 수 있다. HTML5의 〈nav〉 요소와 동일한 역할을 하므로 해당 요소와 중복해서 사용하지 않아야 한다. 또한 〈nav〉 요소처럼 한 페이지 내에서 2~3개 이상 사용하지 않을 것을 권장한다.

만약 여러 개의 navigation role을 사용할 경우 aria-label 속성을 같이 사용하여 어떤 내비게이션인지 이해할 수 있도록 제공하는 것이 필요하다.

• main

메인 콘텐츠 영역을 의미하며 웹 페이지 내에서 main role은 한 번만 선언할 수 있다. main role은 HTML5의 〈main〉 요소와 같은 역할을 하며 〈main〉 요소와 중복해서 사용하지 않도록 해야 한다.

• complementary

메인 콘텐츠를 보충할 수 있는 부가적인 내용을 담는 영역이며 메인 콘텐츠에서 분리되어도 그 자체로 의미가 있는 콘텐츠 영역이다. 통상 메인 콘텐츠의 좌측 또는 우측에 부가정보를 이 영역으로 구분하여 제공할 수 있다. 해당 영역에는 주로 날씨 관련 정보나 주식의 시세 등을 포함하는 경우가 많다. HTML5의 〈aside〉 요소와 동일한 역할이다.

• form

사용자가 입력 가능한 HTML의 〈form〉 영역 임을 의미하며 이때 검색을 위한 폼 영역은 제외하고 지정할 수 있다. 앞서 내용과 마찬가지로 HTML의 〈form〉 요소와 중복 사용은 바람직하지 않다.

• search

검색을 위한 입력 폼 영역을 포함할 수 있다.

• contentinfo

상위 문서의 메타데이터를 담을 수 있는 영역으로 저작권 정보와 주소 및 연락처, 개인정보 정책 등 주로 푸터 콘텐츠로 분류할 수 있는 내용을 포함할 수 있다. HTML5의 〈footer〉 요소와 비슷한 역할을 한다.

5. Live Region

5-1. Live Region

Live Region은 웹 문서나 위젯, 웹 애플리케이션의 전부 또는 일부가 동적으로 변경되는 경우 사용자의 조작 없이 변경된 내용과 진행 상태를 알리기 위해 사용되는 속성으로 모든 요소에 적용할 수 있다.

Live Region에는 업데이트 된 요소를 사용자에게 알려주는 방법을 설정하는 aria-live 속성과, aria-live 속성이 사용된 요소의 DOM이 업데이트 되었을 때 업데이트 된 부분만 알려줄 것인지 전체를 모두 알려줄 것인지를 설정할 수 있는 aria-atomic 속성이 있다.

• aria-live

가장 먼저 aria-live 속성은 기본 값이 "off"이며 이 경우 업데이트 된 정보를 사용자에게 알리지 않는다. aria-live 속성 값으로 "polite"를 지정할 경우 사용자의 입력이 모두 끝나면 그때 업데이트 된 내용을 스크린리더 등의 보조기기 사용자에게 전달하게 되며 "assertive" 값을 지정하면 바로 업데이트 된 내용을 전달할 수 있다.

속성 값	설명
off(기본값)	업데이트 된 내용을 사용자에게 알리지 않음.
polite	사용자의 입력이 끝났을 때 알림.
assertive	업데이트 된 내용이 있을 때 즉시 알림.

• aria-atomic

aria-atomic 속성은 aria-live 속성이 사용된 요소에 "true"와 "false" 값을 지정하여 DOM이 업데이트 되었을 때 업데이트 된 부분만 알려줄 것인지 전체를 모두 알려줄 것인지를 설정할 수 있다.

속성 값	설명
false(기본값)	업데이트 된 내용을 포함하여 전체 내용을 모두 읽음.
true	업데이트 된 내용만 읽음.

```
<div class="container" aria-live="polite" aria-atomic="true">
    ……
</div>
```

• aria-busy

Live Region의 상태 값으로 aria-busy 속성을 지정하면 업데이트가 진행 중인지 여부를 표현할 수 있다. aria-busy="true" 가 설정되면 해당 속성이 제거 되거나, 값이 false으로 변경될 때까지 업데이트되는 내용을 안내하지 않는다.

속성 값	설명
false(기본값)	업데이트 된 내용을 안내 함.
true	업데이트 된 내용이 있음을 안내하지 않음.

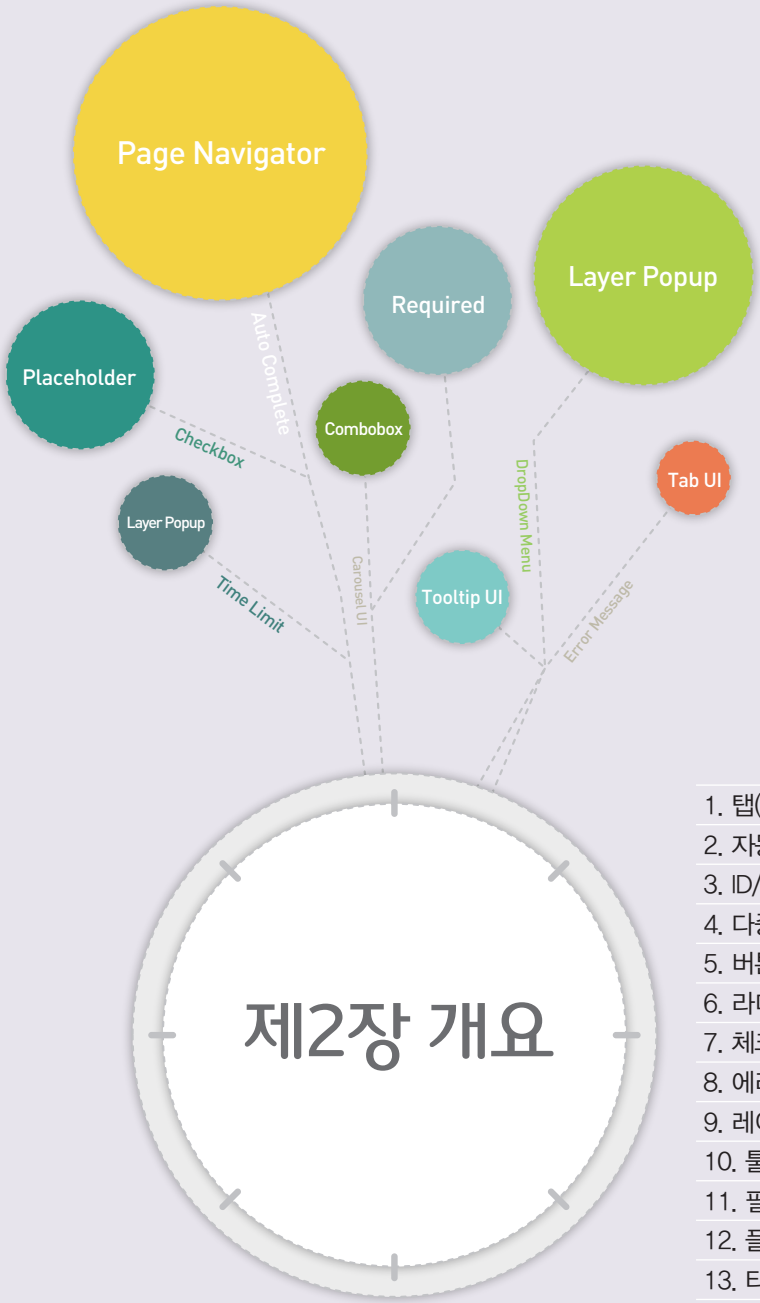
```
<div class="container" aria-live="polite" aria-busy="true">
    ……
</div>
```

• aria- relevant

aria-relevant 속성은 요소 및 텍스트 등의 추가, 삭제 등의 업데이트 정보를 알릴 지 여부를 설정할 수 있다.

속성 값	설명
additions text (기본값)	요소(Element)가 추가되거나 콘텐츠가 변경 되었을 때 안내
additions	요소(Element)가 추가 되었을 때 안내.
removals	요소(Element)가 삭제 되었을 때 안내.
text	콘텐츠가 변경 되었을 때 안내.
all	요소(Element)의 추가, 삭제 및 콘텐츠가 변경되었을 때 안내

```
<div class="container" aria-live="polite" aria-relevant="all">
    ……
</div>
```



제2장 개요

1. 탭(Tab) UI

• 기존 코드의 문제점들

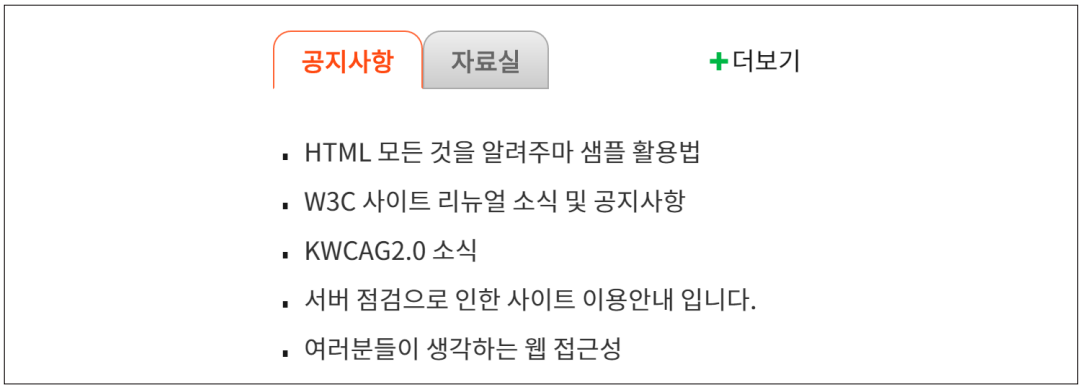
탭 형식의 UI는 웹페이지에서 제공되는 콘텐츠를 보다 편리하게 사용할 수 있도록 해준다. 특히 연관된 정보를 그룹화하여 탭이 선택되었을 때 선별적으로 노출되기 때문에 한 화면에서 보다 많은 콘텐츠를 제공할 수 있다는 장점이 있다.

그러나 이런 탭 형식의 UI를 제공할 경우 전맹, 저시력 등의 스크린리더를 사용하는 사용자에게는 정보 접근에 혼란이 생기거나 정보에 대한 이해가 어려워 질 수 있다.

이번 사례에서는 탭 형식의 UI 사례 분석을 통해 사용자에게 보다 편리한 환경과 향상된 웹 접근성 정보를 제공하는 방법을 학습하기로 한다.

가장 먼저 살펴볼 사례는 웹사이트의 메인 페이지에 자주 등장하는 탭 형식의 게시물 목록이다.

탭 UI를 사용하는 게시물 목록 링크



위와 같은 탭 UI를 제공하기 위해 만약 화면에 보이는 순서 만을 고려한다면 공지사항 탭 ➡ 자료실 탭 ➡ 더보기 링크 ➡ 목록 순으로 생각할 수 있다.

하지만 이 경우 해당 목록이 공지사항에 대한 목록인지 자료실에 대한 목록인지 정보에 대한 이해가 명확하지 않을 수 있다.

화면에 보이는 콘텐츠 형태 만을 고려한 잘못된 사례

```
<div class="tab-interface">
    <ul class="tab-list">
        <li><a href="/">공지사항</a></li>
        <li><a href="/">자료실</a></li>
    </ul>
    <div class="tab-contents">
        <ul class="notice-list">
            <li><a href="/">HTML 모든 것을 알려주마 샘플 활용법</li>
            <li><a href="/">W3C 사이트 리뉴얼 소식 및 공지사항</li>
            <li><a href="/">KWCAG2.0소식</li>
            <li><a href="/">서버 점검으로 인한 사이트 이용안내</li>
            <li><a href="/">여러분들이 생각하는 웹접근성</li>
        </ul>
        <ul class="pds-list">
            <li><a href="/">시각장애인에 닫힌 모바일 앱</li>
            <li><a href="/">개도국 정보화 격차 해소</li>
            <li><a href="/">민간분야 웹 사이트 장애인 접근성 취약</li>
            <li><a href="/">서버 점검으로 인한 사이트 이용안내</li>
            <li><a href="/">웹 창시자 "웹의 권리장전 필요하다"</li>
        </ul>
        <a href="/" class="more">더보기</a>
    </div>
</div>
```

이런 문제점을 해결하고 정보에 대한 올바른 이해를 돕기 위해서는 콘텐츠의 마크업 순서를 공지사항 탭 ➡ 공지사항 목록 ➡ 공지사항 더보기 링크 / 자료실 탭 ➡ 자료실 목록 ➡ 자료실 더보기 링크 순으로 수정해야 한다.

이때 공지사항 탭과 자료실 탭은 해당 콘텐츠 블록의 제목으로 제공한다면 콘텐츠 블록에 대한 명확한 범위를 지정할 수 있고 스크린리더 등의 사용자가 헤딩 간 이동 기능을 사용하여 필요한 정보로 건너뛰기 할 수 있다.

논리적인 순서에 맞게 선형화 된 콘텐츠를 제공한 사례

```
<div class="tab-interface">
    <div class="tab-contents">
        <div id="notice" class="on">
            <h1><a href="/">공지사항</a></h1>
            <ul class="notice-list">
                <li><a href="/">HTML 모든 것을 알려주마 샘플 활용법</li>
                <li><a href="/">W3C 사이트 리뉴얼 소식 및 공지사항</li>
                <li><a href="/">KWCAG2.0소식</li>
                <li><a href="/">서버 점검으로 인한 사이트 이용안내</li>
                <li><a href="/">여러분들이 생각하는 웹 접근성</li>
            </ul>
            <a href="/" class="more" title="공지사항">더보기</a>
        </div>
        <div id="pds">
            <h1><a href="/">자료실</a></h1>
            <ul class="pds-list">
                <li><a href="/">시각장애인에 닫힌 모바일 앱</li>
                <li><a href="/">개도국 정보화 격차 해소</li>
                <li><a href="/">민간 웹 사이트 장애인 접근성 취약</li>
                <li><a href="/">서버 점검으로 인한 사이트 이용안내</li>
                <li><a href="/">웹 창시자 "웹의 권리장전 필요하다"</li>
            </ul>
            <a href="/" class="more" title="자료실">더보기</a>
        </div>
    </div>
</div>
```

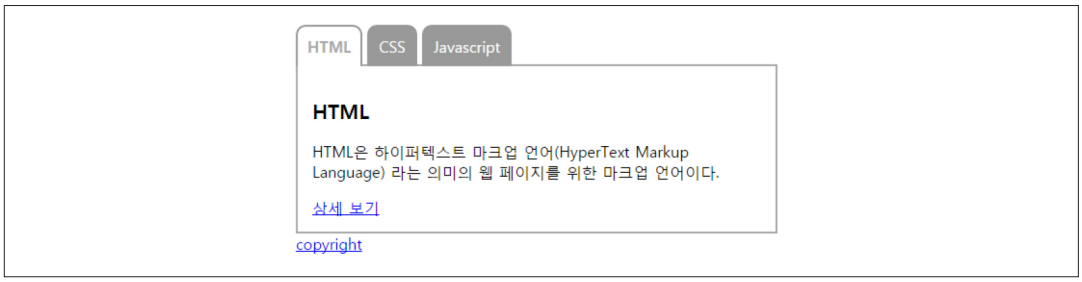
• WAI-ARIA를 사용해야 하는 이유

앞에서 살펴본 사례의 경우 콘텐츠의 마크업을 탭 ➡ 목록 ➡ 더보기 링크 순으로 제공하도록 수정하는 것만으로 스크린리더 사용자에게 탭과 게시물 목록의 연관관계를 이해할 수 있도록 접근성을 개선할 수 있었다. 그러나 현재 활성화 된 탭이 어떤 콘텐츠인지 구분할 수 있는 정보가 없다는 문제점이 남는다.

또 다른 사례로 탭과 많은 양의 본문으로 구성된 형식의 탭 UI라면 마크업 순서를 조정하는 것이 오히려 접근성과 사용성을 해치는 경우가 될 수도 있다.

다음의 사례를 살펴보기로 하자.

본문 섹션 콘텐츠를 탭 UI로 제공한 사례



앞서 살펴본 공지사항 및 자료실 탭 UI와 같이 논리적인 순서로 마크업하기 위해 HTML 탭 ➡ 본문 섹션, CSS 탭 ➡ 본문 섹션, Javascript 탭 ➡ 본문 섹션 순으로 제공한다고 가정해 보자.

이때 스크린리더 사용자나 키보드 사용자는 본문 섹션의 내용이 방대하고 내부에 링크 등 키보드 포커싱이 가능한 콘텐츠가 많을 경우 오히려 탐색에 어려움을 겪을 수 있다. 왜냐하면 마지막 Javascript 탭을 선택하기 위해 앞의 HTML과 CSS 관련 정보를 모두 읽거나 이를 지나쳐 이동해야 하기 때문이다.

또 다른 문제는 스크린리더 사용자에게 HTML, CSS, Javascript 탭을 탭 정보가 아닌 일반적인 링크로만 읽어주게 된다는 점이다.

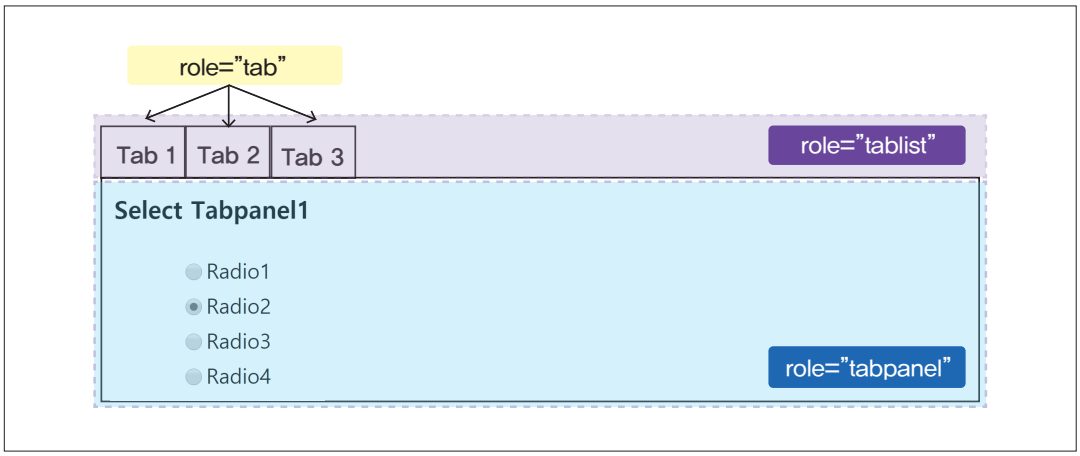
```
<div class="tab-contents">
    <section id="section1">
        <a href="#section1">HTML</a>
        <h1>HTML</h1>
        <p>HTML은 하이퍼텍스트 마크업 언어(HyperText Markup Language)라는 의미의
           웹 페이지를 위한 마크업 언어이다.</p>
        <a href="/">상세보기</a>
    </section>
    <section id="section2">
        <a href="#section2">CSS</a>
        <h1>CSS</h1>
        <p>캐스케이딩 스타일 시트(Cascading Style Sheets, CSS)는 마크업 언어가 실제 표시되는
           방법을 기술하는 언어로, HTML과 XHTML에 주로 쓰이며, XML에서도 사용할 수 있다.</p>
        <a href="/">상세 보기</a>
    </section>
    <section id="section3">
        <a href="#section1">Javascript</a>
        <h1>Javascript</h1>
        <p>자바스크립트(JavaScript)는 객체 기반의 스크립트 프로그래밍 언어이다.
           이 언어는 웹브라우저 내에서 주로 사용하며, 다른 응용 프로그램의 내장 객체에도
           접근할 수 있는 기능을 가지고 있다.</p>
        <a href="/">상세 보기</a>
    </section>
</div>
```

이런 문제점을 해결하기 위해서는 마크업 순서의 조정보다 WAI-ARIA에서 제공되는 role(역할)을 활용하는 것이 바람직하다.

• WAI-ARIA 전체적인 그림과 설명

우선 HTML, CSS, Javascript가 링크가 아닌 탭의 역할을 담당하도록 하고 본문 섹션과 연결을 하기 위해 어떤 마크업 구조가 필요한지 살펴보자.

탭 UI를 위한 WAI-ARIA의 마크업 구조



해당 마크업에 WAI-ARI를 적용하는 방법은 다음과 같다.

- 탭 메뉴를 그룹핑 하는 요소에 tablist role 부여
- 각각의 탭 메뉴 요소에 tab role 부여
- 각 탭이 컨트롤 하는 탭 섹션의 id 값을 aria-controls 속성 값으로 설정
- 각 탭 섹션에 tabpanel role 부여
- 선택된 탭 메뉴에 aria-selected 속성의 값을 true로 설정
- 나머지 탭 메뉴들은 aria-selected 속성의 값을 false로 설정하거나 aria-selected 속성을 제거

• WAI-ARIA 속성 정리

다음은 탭 UI를 제공할 때 필요한 WAI-ARIA의 역할(Role)과 속성(Property)을 정리한 것이다.

요소	역할	속성과 상태	설명
〈ul〉	tablist		탭 집합
〈li〉	presentation		〈li〉 요소의 리스트 의미를 삭제
〈a〉	tab	aria-selected="true/false"	〈a〉 요소의 역할을 탭으로 정의하고, 선택되었을 때 true, 선택되지 않았을 때 false
		aria-contols="{ }"	tabpanel의 id명
		id="{}"	aria-labelledby 속성의 함수명
〈div〉	tabpanel	aria-expanded="true/false"	〈div〉 요소의 역할을 tabpanel로 정의하고, tabpanel이 열렸을 때 true, 닫혔을 때 false
		aria-labelledby="{ }"	tabpanel의 id명
		id="{ }"	aria-controls 속성의 함수명

• 소스 코드 정리

접근성과 사용성 높은 탭 UI를 위해 WAI-ARIA 속성들을 사용하여 구현한 코드는 아래와 같다.

```
<!DOCTYPE html>
<html lang="ko">
    <head>
        <meta charset="utf-8">
        <title>Tab UI</title>
        <style>
             .tab-interface{
                width: 500px;
            }
            .tab-list{
                padding-left: 0;
                margin: 0;
            }
            .tab-list:after{
                display: block;
                clear: both;
                content: "";
            }
            .tab-list li {
                float: left;
                list-style: none;
```

```
        }
        .tab-list li{
            display: block;
            background: #999;
            color: #fff;
            border: 2px solid #999;
            border-bottom: 0;
            padding: 10px;
            margin-right: 5px;
            border-radius: 10px 10px 0 0;
            text-decoration: none;
            position: relative;
            top: 2px;
        }
        [aria-selected="true"]{
            background: #fff !important;
            color: #aaa !important;
            font-weight: bold;
        }
        .tab-contents{
            margin-top: 0;
            padding: 15px;
            border: 2px solid #aaa;
            z-index: 5;
        }
        .tab-contents h1{
            font-size: 1.3em;
        }
        .unvisual{
            display: none;
        }
    </style>
    <script
    src="https:// ajax.googleapis.com/ajax/libs/jquery/1.11.1/jquery.min.js">
    </script>
    <script>
    $(document).ready(function() {
        $('[role="tab"]').keyup(function(e){
            var keyCode = e.keyCode || e.which;// 키보드 코드값
            if(keyCode == 39 || keyCode == 40){// 오른쪽방향키 이거나
                                               // 아래쪽 방향키
```

```
            // 브라우저의 기본 동작을 취소한다.
            e.preventDefault();
            // 다음 tab 요소에 aria-selected=true로 지정하고
            // 형제요소중에 자신 tab 이외의 나머지 tab 요소들을
            // aria-selected=false로 지정한다.
            $(this).next().attr('aria-selected', true)
                   .siblings().attr('aria-selected', false);
            var selectedId = "#"+$(this).next().attr('aria-controls');
            // 자신은 보이게하고 다른 tabpanel은 보이지 않게 지정한다.
            $(selectedId).removeClass('unvisual')
                         .siblings().addClass('unvisual');
            // 다음요소로 포커스를 이동한다.
            $(this).next().focus();
            // 마지막요소에서 오른쪽 방향키나 아래쪽 방향키를 눌렀을 경우
            if($(this).next()
                      .prevObject.attr('aria-controls')=='section3'){
                // tab, tabpanel, focus 모두 처음으로 이동
                $('#tab1').attr('aria-selected', true)
                          .siblings().attr('aria-selected', false);
                $('#section1').removeClass('unvisual')
                              .siblings().addClass('unvisual');
                $('#tab1').focus();
            }
        }
        if(keyCode == 37 || keyCode ==38){// 왼쪽방향키 이거나
                                          // 위쪽 방향키
            e.preventDefault();
            // 이전 tab 요소에 aria-selected=true로 지정하고
            // 형제요소중에 자신 tab 이외의 나머지 tab 요소들을
            // aria-selected=false로 지정한다.
            $(this).prev().attr('aria-selected', true)
                   .siblings().attr('aria-selected', false);
            var selectedId = "#"+$(this).prev().attr('aria-controls');
            // 자신은 보이게하고 다른 tabpanel은 보이지 않게 지정한다.
            $(selectedId).removeClass('unvisual')
                         .siblings().addClass('unvisual');
            // 이전요소로 포커스를 이동한다.
            $(this).prev().focus();
            // 처음요소에서 왼쪽 방향키나 위쪽 방향키를 눌렀을 경우
            if($(this).prev().prevObject
                      .attr('aria-controls')=='section1'){
```

```
                // tab, tabpanel, focus 모두 마지막으로 이동
                $('#tab3').attr('aria-selected', true)
                          .siblings().attr('aria-selected', false);
                $('#section3').removeClass('unvisual')
                              .siblings().addClass('unvisual')
                $('#tab3').focus();
            }
        }
        if(keyCode == 35){// end 키를 눌렀을 때
            e.preventDefault();
            // tab, tabpanel, focus 모두 마지막으로 이동
            $('#tab3').attr('aria-selected', true)
                      .siblings().attr('aria-selected', false);
            $('#section3').removeClass('unvisual')
                          .siblings().addClass('unvisual');
            $('#tab3').focus();
        }
        if(keyCode == 36){// home키를 눌렀을 때
            e.preventDefault();
            // tab, tabpanel, focus 모두 처음으로 이동
            $('#tab1').attr('aria-selected', true)
                      .siblings().attr('aria-selected', false);
            $('#section1').removeClass('unvisual')
                          .siblings().addClass('unvisual');
            $('#tab1').focus();
        }
    });
    $('[role="tab"]').keydown(function(e){
        var keyCode = e.keyCode || e.which;// 키보드 코드값
        if(keyCode == 9){// 탭키를 눌렀을 때
            e.preventDefault();
            var selectedId = "#"+$(this).attr('aria-controls');
            console.log(selectedId);
            $(selectedId).children('a').focus();
        }
    });
    $('section a').keydown(function(e){
        var keyCode = e.keyCode || e.which;// 키보드 코드값
        if (keyCode == 9 && e.shiftKey) {// shift+tab 키
            $('.tab-list li').each(function( index ) {
                if($( this ).attr('aria-selected') == 'true'){
```

```
                    $( this ).next().focus();
                    return false;
                }
            });
        }
    });
    // tab 요소에 클릭 이벤트를 추가한다.
    $('[role="tab"]').on('click', function(e) {
        e.preventDefault();
        // 클릭한 tab 요소에 aria-selected=true로 지정하고
        // 형제요소중에 자신 tab 이외의 나머지 tab 요소들을
        // aria-selected=false로 지정한다.
        $(this).attr('aria-selected', true)
               .siblings().attr('aria-selected', false);
        var selectedId = "#"+$(this).attr('aria-controls');
        // 자신은 보이게하고 다른 tabpanel은 보이지 않게 지정한다.
        $(selectedId).removeClass('unvisual')
                     .siblings().addClass('unvisual');
    });
});
</script>
</head>
<body>
<div class="tab-interface">
    <!-- 탭 인덱스 -->
    <ul class="tab-list" role="tablist">
        <li id="tab1" role="tab" aria-controls="section1"
            aria-selected="true" tabindex="0">HTML</li>
        <li id="tab2" role="tab" aria-controls="section2"
            aria-selected="false" tabindex="0">CSS</li>
        <li id="tab3" role="tab" aria-controls="section3"
            aria-selected="false" tabindex="0">Javascrip</li>
    </ul>
    <!-- 탭 콘텐츠 -->
    <div class="tab-contents">
        <section id="section1" role="tabpanel" aria-labelledby="tab1">
            <p>
                HTML은 하이퍼텍스트 마크업 언어(HyperText Markup Language)
                라는 의미의 웹 페이지를 위한 마크업 언어이다.
            </p>
            <a href="#">상세 보기</a>
```

```
            </section>
            <section id="section2" class="unvisual" role="tabpanel"
                     aria-labelledby="tab2">
                <p>
                    캐스케이딩 스타일 시트(Cascading Style Sheets, CSS)는
                    마크업 언어가 실제 표시되는 방법을 기술하는 언어로, HTML과 XHTML에
                    주로 쓰이며, XML에서도 사용할 수 있다.
                </p>
                <a href="#">상세 보기</a>
            </section>
            <section id="section3" class="unvisual" role="tabpanel"
                     aria-labelledby="tab3">
                <p>
                    자바스크립트(JavaScript)는 객체 기반의 스크립트 프로그래밍
                    언어이다. 이 언어는 웹브라우저 내에서 주로 사용하며,
                    다른 응용 프로그램의 내장 객체에도 접근할 수 있는 기능을 가지고 있다.
                </p>
                <a href="#">상세 보기</a>
            </section>
        </div>
    </div>
    <a href="#">copyright</a>
    </body>
</html>
```

• 키보드 인터랙션

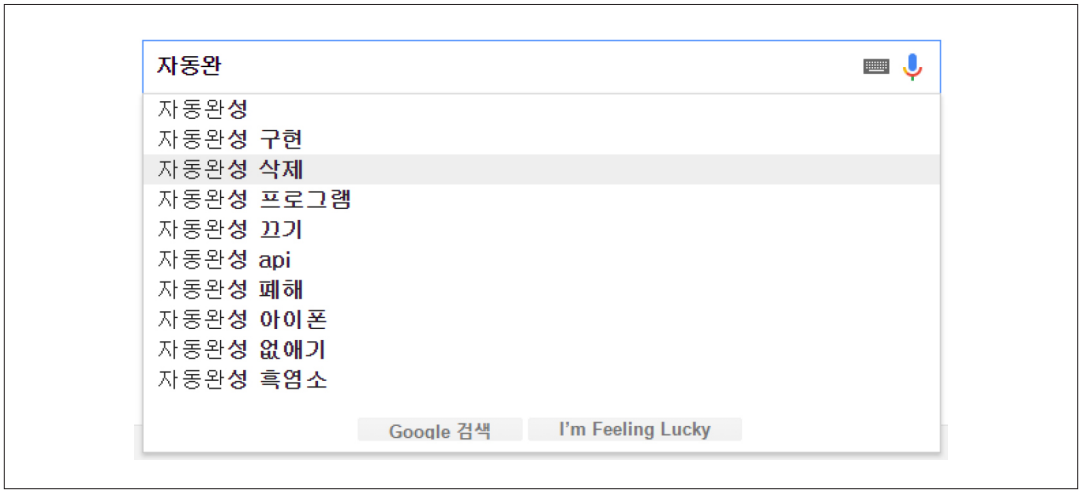
키보드	인터랙션
왼쪽/위쪽 방향키	이전 탭으로 이동, 처음 탭일 경우는 마지막 탭으로 이동
오른쪽/아래쪽 방향키	다음 탭으로 이동, 마지막 탭일 경우는 처음 탭으로 이동
탭키	활성화 된 순서대로 이동 (비활성화 탭으로 이동하지 않는다)
Home키	처음 탭으로 이동
End키	마지막 탭으로 이동

2. 자동 완성(Auto Complete) UI

• 기존 코드의 문제점들

자동 완성(Auto Complete) 기능은 주로 검색 기능, SNS의 참조 친구 목록, tag 목록 등에서 쉽게 찾아볼 수 있는데, 사용자의 입력 값에 따라 해당 값이 포함된 몇 가지 자동 완성어를 제안해 주어 사용자가 "선택"하는 행동만으로 쉽게 사용할 수 있도록 제공하는데 사용된다.

구글 검색에 사용된 자동 완성



위와 같은 자동 완성(Auto Complete) UI가 제공되고 있는 국내 사이트의 코드를 하나 보도록 하자.
(코드를 좀 더 간결하게 볼 수 있도록 일부 코드는 생략 되었다.)

A 사이트 자동 완성 UI 마크업

```
<div>
    <form action="..." id="search" method="get" name="search">
        <fieldset>
            <legend>통합 검색</legend>
            <input type="text" autocomplete="off" id="keyword"
                   name="keyword">
            <button type="submit">검색</button>
        </fieldset>
    </form>
</div>
<ul id="autocomplete">
    <li>
        <a href="#" onclick="search('라면')">
            <span class="term"><strong>라면</strong></span>
        </a>
    </li>
    <li>
        <a href="#" onclick="search('컵라면')">
            <span class="term">컵<strong>라면</strong></span>
        </a>
    </li>
    <li>
        <a href="#" onclick="search('신라면')">
            <span class="term">신<strong>라면</strong></span>
        </a>
    </li>
    <li>
        <a href="#" onclick="search('진라면')">
            <span class="term">진<strong>라면</strong></span>
        </a>
    </li>
</ul>
```

코드 자체만을 보았을 때, 문서의 구조 혹은 순서 상으로는 별달리 문제가 없어 보이고 일반 사용자에게는 크게 문제가 없이 제공될 수 있을 것으로 예상할 수 있다.

하지만 위 경우 스크린리더 사용자에게는 입력 상자는 단순히 입력 상자일 뿐이고, 사용자가 입력한 검색어에 대한 추천 검색어 목록이 제공된다 하더라도 사용자로서는 이 목록의 제공 여부를 알 수가 없는 문제가 발생된다.

따라서 스크린리더 사용자는 처음부터 추천 검색어를 활용할 생각 조차 할 수 없게 되므로, 비 스크린리더 사용자가 사용하는 것과 달리 “선택을 위한 탐색”의 행위를 하지 않게 된다.

• WAI-ARIA를 사용해야 하는 이유

앞서 본 코드의 경우 스크린리더 사용자에게는 입력상자와 추천 검색어 목록 사이에 의미적으로도, 기능적으로도 어떠한 연관성을 가지지 못하고 있다.

하지만, 여기에 combobox 역할(Role)을 추가함으로서 해당 입력상자가 콤보박스(combobox)임을 인지할 수 있어 스크린리더 사용자로 하여금 어떤 방법으로 사용할 수 있는지를 인지할 수 있게 된다.

해당 추천 검색 목록에 listbox 역할(Role)을 추가하고, 각 항목에 option 역할(Role)을, 항목의 선택에 따라 aria-activedescendant 속성(Property)을 적용하게 되면 사용자는 현재 선택된 항목이 어떤 것인지를 명확하게 인지 할 수 있게 된다.

또한, 추천 검색 목록의 제공 여부와 몇 개의 검색 목록이 제공되었는지 상태를 실시간으로 알려주기 위해 status 역할(Role)을 적용하여 여기에 상태 정보를 추가해주면, 사용자는 실시간으로 업데이트 되는 추천 검색어 목록이 제공 여부와 개수를 알 수 있게 된다.

따라서 WAI-ARIA의 적용은 사용자에게 보다 향상된 접근성을 제공할 수 있는 계기가 될 것으로 기대할 수 있다.

• WAI-ARIA 전체적인 그림과 설명

우선 전체적인 WAI-ARIA의 역할(Role), 속성(Property), 상태(State) 구조를 그림으로 살펴보자.

먼저, 〈input〉 요소에 combobox 역할(Role)을 추가하고, 자동완성 기능이 목록 형태로 지원 됨을 알 수 있도록 aria-autocomplete 속성(Property)을 list로 설정한다. 또한 추천 검색어 목록이 하위 메뉴 형태로 존재함을 인식할 수 있도록 aria-haspopup 속성(Property)을 true로 둔다.

WAI-ARIA 역할(Role), 속성(Property), 상태(State) 구조


```
<input type="text" name="search" id="search" autocomplete="off"
       role="combobox" aria-autocomplete="list" aria-haspopup="true">
```

추천 검색어 목록이 제공될 때, 추천 검색어의 존재 여부와 제공 된 추천 검색어의 개수를 전달 할 수 있도록, 상태 정보를 나타낼 요소(Element)에 status 역할(Role)을 추가한다.

status 역할(Role)은 WAI-ARIA Live Region이기 때문에 aria-live 속성을 사용할 필요가 없기는 하지만, 이를 완전히 지원하지 못하는 경우가 있기 때문에 최대 호환성 제공을 위해 aria-live 속성(Property)을 polite로, aria-relevant 속성(Property)을 additions로 설정한다.

참고로, 추천 검색어의 개수를 나타내는 텍스트는 스크린리더 사용자에게 중요한 정보인 개수를 앞에 두어 사용자에게 전달해야 할 핵심 정보를 먼저 인지할 수 있도록 제공한다.

```
<div id="status"
     role="status" aria-live="polite" aria-relevant="additions">
    <div>10개의 추천 검색어가 있습니다.</div>
</div>
```

추천 검색어 목록으로 사용되는 요소에 listbox 역할(Role)을 추가하여 주고, 각 요소의 항목에 해당하는 요소에는 option 역할(Role)을 추가하고 각 항목별 id값을 부여한다.

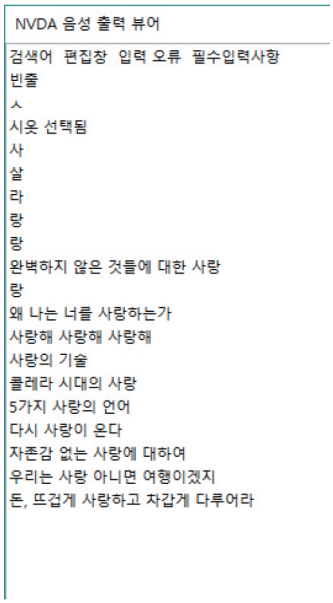
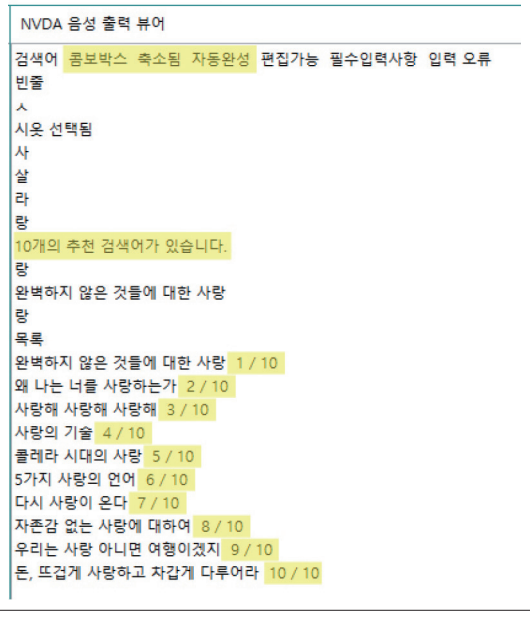
```
<ul role="listbox" >
    <li id="list0" role="option" >완벽하지 않은 것들에 대한 사랑</li>
    <li id="list1" role="option">사랑해 사랑해 사랑해</li>
    <li id="list2" role="option">나를 사랑하지 않는 나에게</li>
    <li id="list3" role="option">왜 나는 너를 사랑하는가</li>
    <li id="list4" role="option">사랑해요 엄마</li>
    <li id="list5" role="option">미안하고 고맙고 사랑해</li>
    <li id="list6" role="option">사랑의 기술</li>
    <li id="list7" role="option">5가지 사랑의 언어</li>
    <li id="list8" role="option">사랑하는 내 아이에게 갓 구운 빵과 쿠키</li>
    <li id="list9" role="option">모네가 사랑한 정원</li>
</ul>
```

사용자가 키보드 인터랙션으로 추천 검색어 목록을 하나씩 "탐색"하게 되면, 각 항목들의 ID 값을 combobox 역할(Role)을 가진 〈input〉 요소(Element)에 aria-activedescendant 속성(Property)의 값으로 업데이트 하고, value값을 그 항목의 텍스트로 설정해 준다.

```
<input type="text" name="search" id="search" autocomplete="off"
       role="combobox" aria-autocomplete="list" aria-haspopup="true"
       aria-activedescendant="list1"  >
<ul role="listbox">
    <li id="list0" role="option">완벽하지 않은 것들에 대한 사랑</li>
    <li id="list1" role="option" class="active">사랑해 사랑해 사랑해</li>
    <li id="list2" role="option">나를 사랑하지 않는 나에게</li>
    <li id="list3" role="option">왜 나는 너를 사랑하는가</li>
    <li id="list4" role="option">사랑해요 엄마</li>
    <li id="list5" role="option">미안하고 고맙고 사랑해</li>
    <li id="list6" role="option">사랑의 기술</li>
    <li id="list7" role="option">5가지 사랑의 언어</li>
    <li id="list8" role="option">사랑하는 내 아이에게 갓 구운 빵과 쿠키</li>
    <li id="list9" role="option">모네가 사랑한 정원</li>
</ul>
```

이렇게 WAI-ARIA를 적용했을 경우 스크린리더가 다음과 같이 알려줄 수 있게 된다.

WAI-ARIA 적용 이전/이후 NVDA음성 출력 내용

WAI-ARIA 적용 이전	WAI-ARIA 적용 이전
	

• WAI-ARIA 속성 정리

각 요소에 적용할 역할(Role)과 속성(Property), 상태(State)를 정리해보면 다음과 같다.

요소	역할	속성/상태	설명
〈input〉	combobox		콤보박스 역할을 정의
		aria-haspopup	텍스트 필드와 연관된 하위 수준의 메뉴가 있으면 true, 그렇지 않으면 false
		aria-autocomplete	사용자 입력에 대한 자동완성 지원 여부를 설정
		aria- activedescendant	활성화 된 제안 목록으로 연결 사용자 인터랙션에 따라 실시간으로 업데이트
		aria-expanded	펼침 상태 설정
〈div〉	status		상태를 표시하는 역할 설정
		aria-live	실시간 알림 속성 설정
		aria-relevant	상태의 업데이트 알림 내용 설정
〈ul〉	listbox		목록에서 하나 이상의 항목을 선택할 수 있는 역할로 설정
〈li〉	option		선택 목록의 항목 역할로 설정

• 소스 코드 정리

이제, 앞서 설명한 자동 완성(Auto Complete) UI 기능을 만들어 보자.

먼저, 자바스크립트에 의해 동작하는 기능과 관련된 WAI-ARIA 역할(Role), 속성(Property), 상태(State)를 제외한 부분만을 마크업 한다.

```
<form action="#" name="search" id="search" method="get">
    <fieldset id="autocomplete">
        <label class="hidden-accessible" for="keyword">검색어</label>
        <input type="text" name="keyword" id="keyword" required >
        <button type="submit">검색</button>
    </fieldset>
</form>
```

작성될 자바스크립트는 jQuery 라이브러리 형태로 플러그인화 할 것이나, 설명을 위해 본문에서는 절차적으로 진행하며 파이어폭스에서의 한글 입력 시 keyup 이벤트 동작 이슈 등의 부가적인 사항들은 제외하고 진행한다.

먼저, 자동 완성(Auto Complete) UI를 설정할 대상 요소, combobox로 설정할 〈input〉 요소를 찾아 변수에 참조시켜두고, 추천 키워드, 상태 정보를 업데이트할 요소를 생성하여 변수에 참조시킨다.

이어서 사용자가 키보드 내비게이팅 하기 전 입력 값을 저장해 둘 변수를 설정한다.

```
var $autocomplete = $('#autocomplete'),
    // combobox로 설정할 input 요소를 참조
    $txtField = $autocomplete.find('input:text'),
    // 추천 키워드 wrapper 요소 참조
    $suggestedWrap = $('<div />'),
    // 상태 정보를 업데이트할 요소 참조
    $status = $('<div class="hidden-accessible" />'),
    // 사용자의 키보드 내비게이팅 이전 값을 저장해 두기 위한 변수
    orgKeyword = '',
    // 사용자의 키 입력 중 수시로 추천 검색어를 가져오는 것을 방지하기 위한 타이머
    delayTimer = null;
```

이후, 〈input〉 요소에 WAI-ARIA 속성들을 설정하고, 브라우저가 자체적으로 지원하는 자동 완성 기능을 꺼두도록 off로 설정한다. 또한 추천 키워드를 담아둘 요소 $suggestedWrap, 상태 정보를 업데이트 할 요소 $status에 WAI-ARIA 속성들을 설정하여 해당 컴포넌트 요소 $autocomplete에 추가한다.

```
// WAI-ARIA Role, Property, State 초기화
$txtField.attr({
    'role' : 'combobox',
    'aria-haspopup' : 'true',
    'aria-autocomplete' : 'list',
    'autocomplete' : 'off'
});
$suggestedWrap.appendTo($autocomplete);
$status.attr({
    'role' : 'status',
    'aria-live' : 'polite',
    'aria-relevant' : 'additions'
}).appendTo($autocomplete);
```

위 자바스크립트가 실행되면, 다음과 같이 동적으로 코드가 생성된다.

```
<fieldset id="autocomplete">
    <label class="hidden-accessible" for="keyword">검색어</label>
    <input type="text" name="keyword" id="keyword" required
           role="combobox" aria-haspopup="true" aria-autocomplete="list"
           autocomplete="off">
    <button type="submit">검색</button>
    <div></div>
    <div class="hidden-accessible"
         role="status" aria-live="polite" aria-relevant="additions">
    </div>
</fieldset>
```

이어서 사용자가 $txtField에 검색 키워드를 입력할 때 Ajax 호출을 통해 새로운 데이터를 얻어와 추천 검색어 목록을 업데이트 하는 함수 updatList와, 추천 검색어 목록을 탐색, 선택하기 위한 기능을 위한 함수 bindKeyEvent를 각각 keyup, keydown 이벤트에 대한 이벤트 핸들러로 바인딩 한다.

```
// $txtField 요소에 keyboard 이벤트 핸들러 바인딩
$txtField.on({
    keyup : updateList,
    keydown : bindKeyEvent
});
```

updateList 함수는 입력 상자에 입력되는 키워드에 대응시켜야 하므로 키보드 내비게이팅에 필요한 키 이외의 키에만 대응하도록 한다.

단, 검색어가 입력될 때마다 API를 호출하게 된다면 과도한 호출이 발생 되기 때문에 일정 시간 동안 키 입력이 없을 경우에 API를 호출 하도록 처리한다.

```
/**
 * @function updateList
 * @param {event} event
 */
function updateList(event){
    event = event || window.event;
    event.stopImmediatePropagation();
    var keycode = event.which || event.keyCode;
    var keyword = $txtField.val();
    if( orgKeyword === keyword ){
        return;
    }
    switch(keycode){
        case 13 :
        case 27 :
        case 38 :
        case 40 :
            event.preventDefault ? event.preventDefault() :
event.returnValue = false;
            break;
        default :
            orgKeyword = keyword;
            // 일정 시간 동안 키 입력이 없을 때 API 호출
            clearTimeout(delayTimer);
            delayTimer = setTimeout(function(){
                if( keyword === '' ){
                    removeList();
                    return;
                }
                callAPI(keyword);
            }, 400);
    }
}
```

실질적으로 API를 호출하는 callAPI는 사용자가 입력한 키워드를 인자로 전달받아 처리하도록 한다.

이 예제에서는 DAUM의 책 검색 API를 활용했다.

```
/**
 * @function callAPI
 * @param {string} keyword
 */
function callAPI(keyword){
    $.ajax({
        url : 'https:// apis.daum.net/search/book',
        data : {
            apikey : '...',
            output : 'json',
            display : 15,
            searchType : 'title',
            sort : 'accu',
            q : keyword
        },
        method : 'GET',
        dataType : 'jsonp',
        cache : false
    })
    .done(function(data){
        // JSON 데이터 가공 (검색어로 쓸 데이터만 추출)
        for (var i = -1, source = [], item = null;
                item = data.channel.item[++i] ; ){
            source.push(
                item['title'].replace(/\&lt;b\&gt;|\&lt;\/b\&gt;/g, '')
            )
        }
        // 추출된 검색어 목록으로 검색 목록을 렌더링
        renderList(source);
    });
}
```

renderList 함수는 추천 검색어 목록을 생성/업데이트 하며 동시에 상태 정보를 업데이트 하는 역할을 담당한다.

여기서 생성되는 목록 〈ul〉 요소에는 앞서 설명한 대로 listbox 역할(Role)을 설정하고, 〈li〉 요소에는 option 역할(Role)을 설정하도록 한다. 아울러, 현재 업데이트 된 항목의 개수를 다른 함수들에서도 쉽게 참조 할 수 있도록 〈ul〉 요소에 data–* 속성(Attribute)을 이용하여 추가한다.

상태 정보를 담을 $state에는 "n개의 추천 검색어가 있습니다"라는 텍스트를 가진 〈div〉 요소를 생성하여 추가하도록 한다.
(여기서 주의 할 점은 추가되는 요소가 Phrasing 콘텐츠 모델인 요소 – 예를 들어 〈span〉 요소인 경우 일부 스크린리더에서 노드 추가 상황을 인식하지 못하는 이슈가 있으므로 가급적 Flow 콘텐츠 모델의 요소를 사용하되 자체적으로 나타내는 의미가 없는 〈div〉 요소를 사용하는 것을 권장한다.)

```
/**
 * @function renderList
 * @param {array} source
 */
var $suggestedList = null;
function renderList(source){
    if($suggestedList === null){
        // 추천 검색어 목록 생성
        $suggestedList = $('<ul class="listbox" />');
        // 추천 검색어 목록 초기화
        // WAI-ARIA 적용, 이벤트 바인딩
        $suggestedList
        .attr({
            'role' : 'listbox',
            'data-count' : source.length
        })
        .appendTo($suggestedWrap)
        .on({
            'mousedown' : function(event){
                // 마우스로 선택 시 처리
                event = event || window.event;
                event.stopPropagation();
                var activatedItem = $txtField.attr('aria-activedescendant');
                deSelectItem($('#' + activatedItem).index());
                selectItem($(this).index());
            }
        }, 'li');
        // 추천 검색어 목록 외 영역 클릭시 초기화 처리
        $(document).on({
```

```
            mousedown : function(event){
                event = event || window.event;
                if(event.target === $txtField[0]){
                    return;
                }
                removeList();
            }
        });
    }

    // source로부터 추천 검색어 항목 생성
    var docFrag = document.createDocumentFragment();
    for(var i = -1, item = null; item = source[++i];){
        $('<li />')
            .attr({'id' : 'item' + i, 'role' : 'option'})
            .text(item)
            .appendTo(docFrag);
    }
    $suggestedList
        .attr({
            'data-count' : source.length
        })
        .empty()
        .append(docFrag);
    // 상태 정보 업데이트
    $txtField.removeAttr('aria-activedescendant').attr({'aria-expanded' :
'true'});
    var state = $('<div />').text(source.length + '개의 추천 검색어가 있습니다.');
    $status.empty().append(state);
}
```

위 코드의 실행 결과로 추천 검색어 목록이 $state에 추가되게 되고, 다음과 같이 코드가 동적으로 생성된다.

```
<div>
    <ul class="listbox" role="listbox" data-count="10">
        <li id="item0" role="option">완벽하지 않은 것들에 대한 사랑</li>
        <li id="item1" role="option">왜 나는 너를 사랑하는가</li>
        <li id="item2" role="option">사랑해 사랑해 사랑해</li>
        <li id="item3" role="option">사랑의 기술</li>
        <li id="item4" role="option">콜레라 시대의 사랑</li>
        <li id="item5" role="option">5가지 사랑의 언어</li>
        <li id="item6" role="option">다시 사랑이 온다</li>
        <li id="item7" role="option">자존감 없는 사랑에 대하여</li>
        <li id="item8" role="option">우리는 사랑 아니면 여행이겠지</li>
        <li id="item9" role="option">돈, 뜨겁게 사랑하고 차갑게 다루어라</li>
    </ul>
</div>
<div class="hidden-accessible"
     role="status" aria-live="polite" aria-relevant="additions">
    <div>10개의 추천 검색어가 있습니다.</div>
</div>
```

다음으로, 사용자 키보드 인터랙션을 담당하는 bindKeyEvent 함수를 만들어 보자.

bindKeyEvent 함수는 상하 방향키, Esc키, 엔터키에 반응하도록 하며, 다음과 같은 인터랙션이 이루어지도록 한다.

- 추천 검색어 목록이 생성 된 직후 아래쪽 방향키를 눌렀을 경우 추천 검색어 목록의 첫 번째 목록을 선택
- 추천 검색어 목록에 선택된 항목이 있을 경우 각 항목을 상하로 이동하여 선택항목을 변경 (기존 것의 선택을 해제하도록 deSelectItem를 호출하고, 새로운 것을 선택하도록 selectItem 함수를 호출)
- 첫 번째 항목이 선택된 상황에서 위쪽 방향키를 눌렀을 경우, 마지막 항목이 선택된 상황에서 아래쪽 방향키를 눌렀을 경우, Esc키를 눌렀을 경우, 추천 검색어 목록을 제거하고 처음 사용자가 입력했던 값으로 초기화.
- 엔터키를 눌렀을 경우 입력된 값을 가지고 해당 양식을 전송
  (단, 입력값이 없거나 추천 검색어 목록이 없는 경우 사용자가 입력한 검색어를 가지고 해당 양식을 전송)

```
/**
 * @function bindKeyEvent
 * @param {event} event
 */
function bindKeyEvent(event){
    event = event || window.event;
    event.stopImmediatePropagation();
    var keycode = event.keyCode || event.which,
        idx = 0,
        activatedItem = $txtField.attr('aria-activedescendant');
    switch(keycode){
        case 13 :   // enter
            // 선택된 항목이 있는 경우
            if(activatedItem !== undefined){
                event.preventDefault ? event.preventDefault() :
event.returnValue = false;
                $txtField.val($('#' + activatedItem).text());
                $txtField.parents('form').submit();
            }else{
                $txtField.parents('form').submit();
            }
            break;
        case 27 :   // Esc
            event.preventDefault ? event.preventDefault() : event.returnValue =
false;
            $txtField.val(orgKeyword);
            removeList();
            break;
        case 38 :   // up arrow key
            event.preventDefault ? event.preventDefault() : event.returnValue =
false;
            if($suggestedList === null ){
                return;
            }
            if( $txtField.attr('aria-activedescendant') === undefined ){
                idx = -1;
            }else{
                idx = $txtField
                        .attr('aria-activedescendant')
                        .replace('item', '') * 1 - 1;
            }
            deSelectItem(idx + 1);
            selectItem(idx);
            break;
        case 40 :   // down arrow key
            event.preventDefault ? event.preventDefault() : event.returnValue =
false;
```

```
            if($suggestedList === null ){
                return;
            }
            if( $txtField.attr('aria-activedescendant') === undefined ){
                orgKeyword = $txtField.val();
                idx = 0;
            }else{
                idx = $txtField.attr('aria-activedescendant')
                    .replace('item', '') * 1 + 1;
            }
            deSelectItem(idx - 1);
            selectItem(idx);
            break;
    }
}
```

selectItem 함수는 선택 되었다라는 가시성을 부여하기 위해 class(예제에서는 active)를 추가하고, 입력 상자 $txtField에 aria-activedescendant 속성(Property)을 선택 된 항목의 id 값으로 설정하며 value 값을 해당 항목의 텍스트로 설정해 준다.

deSelectItem 함수는 단순히 가시성만을 부여(선택 되었다라는 표시를 해제)하도록 한다.

```
/**
 * @function selectItem
 * @param {number} idx the index number of Item that will be selected
 */
function selectItem(idx){
    if(idx < 0 || $suggestedList === null || idx >=
$suggestedList.data('count') ){
        $txtField.val(orgKeyword);
        removeList();
        return;
    }
    var value = $suggestedList
                    .children('li:eq('+ idx +')')
                    .addClass('active')
                    .text();
    $txtField
        .attr({
            'aria-activedescendant' : 'item' + idx
        })
        .val(value);
}
```

위 코드에 의해 〈input〉 요소에 동적으로 aria-activedescendant 속성(Property)의 값이 업데이트 되어 다음과 같은 코드 결과가 나타나게 되고, 스크린리더는 사용자에게 해당 요소가 선택되었음을 알려주게 된다.

```
<input type="text" name="search" id="search"
       aria-describedby="autoCompleteGuide"
       role="combobox" aria-haspopup="true" aria-autocomplete="list"
       autocomplete="off" aria-activedescendant="aria-item0">
```

다음은 선택을 해제하는 deSelect 함수의 코드이다. 선택 해제는 단순히 가시성만을 제거하면 되기 때문에 selectItem에 의해 추가된 클래스를 제거해 주는 것만으로 충분하다.

```
/**
 * @function deSelectItem
 * @param {number} idx the index number of Item that will be deselected
 */
function deSelectItem(idx){
    $suggestedList.children('li:eq('+ idx +')').removeClass('active');
}
```

마지막으로 추천 검색어 목록을 DOM에서 제거하는 removeList 함수는 다음과 같다.

이 함수는 추천 검색어 목록 $suggestedList를 DOM에서 제거하고, null값으로 초기화 하며, 입력 상자 $txtField에 설정되었던 aria-activedecsendant 속성(Property)을 제거한 뒤, 입력 상자의 value 값을 처음 입력되었던 값으로 초기화 한다 또한, 목록이 닫혔으므로 입력 상자에 aria-expanded 상태(State)를 false로 설정한다.

```
/**
 * @function removeList
 */
function removeList(){
    if( $suggestedList !== null){
        $suggestedList.remove();
        $suggestedList = null;
        $txtField.removeAttr('aria-activedescendant')
                 .attr({'aria-expanded' : 'false'});
        $status.empty();
    }
}
```

• 키보드 인터랙션

구현된 UI의 사용자 키보드 인터랙션을 정리해보면 다음과 같다.

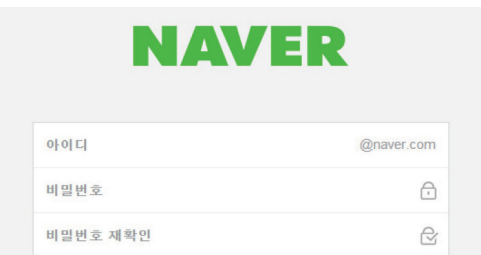
키보드	인터랙션
상하 방향키	추천 검색어가 있는 경우 이전/다음 항목을 선택한다.
엔터키	현재 선택된 항목의 텍스트를 가지고 양식을 전송한다. 만일 선택된 항목이 없거나 제공된 추천 검색어가 없을 시 사용자가 입력한 키워드를 가지고 양식을 전송한다.
Esc키	추천 검색어 목록을 닫고 문서에서 제거한다.

3. ID/PASSWORD 입력서식

• 기존 코드의 문제점들

국내에서 가장 큰 포털 사이트의 NAVER의 회원가입 페이지이다. 아이디를 생성하는 입력상자에 어떻게 입력해야 하는지 알 수가 없고, 잘못 입력하고 발생하는 에러 메시지를 통해서만 "5~20자의 영문 소문자, 숫자와 특수기호(_), (-)만 사용 가능합니다."라는 것을 알 수 있게 된다. 또한 비밀번호도 오류가 발생한 후에 입력도움을 알 수 있기 때문에 스크린리더 사용자는 상당한 불편을 겪게 된다.

NAVER의 아이디와 패스워드 입력박스

입력 전 화면	입력 후 화면
	

스크린리더(NVDA) 음성

- 아이디 편집창
- 비밀번호 편집창

NAVER와 같이 “5~20자의 영문 소문자, 숫자와 특수기호(_), (-)만 사용 가능합니다.”라는 입력 지침을 에러메시지를 통해서만 제공하는 방법은 스크린리더 사용자에게 좋지 않은 경험이 된다.

• WAI-ARIA를 사용해야 하는 이유

“에러메시지”의 잘못된 예와 같이, NAVER 회원가입 페이지에서는 에러가 발생하여도 스크린리더 사용자는 에러가 발생 된 상황을 전혀 알 수 없기 때문에 “1111”이라고 아이디를 입력했지만, NAVER가 정해놓은 입력 기준에 맞지 않는지는 알 수가 없다.

이때 입력 서식인 〈input〉 요소에 aria-label 또는 aria-labelledby와 aria-describedby 속성을 사용하게 되면, 이 문제를 간단히 해결할 수 있다.

• WAI-ARIA 전체적인 그림과 설명

〈input〉 요소을 사용하면 당연히 〈label〉 요소을 사용해야 하는 것이 원칙이지만, 여기에서는 제목 역할을 〈label〉 요소 대신 aria-label 속성으로 사용하려고 한다.

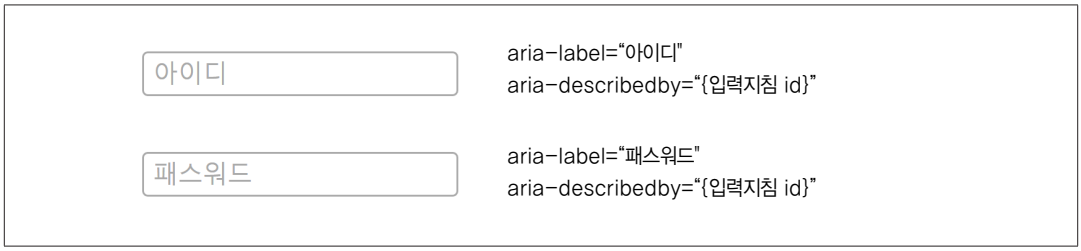
Note.

〈input〉 요소에 aria-label 속성이나 aria-labelledby 속성을 사용하면 〈label〉 요소의 텍스트는 읽지 않고, 〈button〉 요소에 사용하면 〈button〉 요소의 텍스트를 읽지 않으므로 주의해야 한다. (단, 〈input〉 요소의 title 속성은 읽는다.)

〈label〉 요소의 텍스트를 읽어야 하는 경우는 aria-labelledby 속성 대신 aria-describedby 속성을 사용하면 된다. 그러면 스크린리더는 〈label〉 요소의 텍스트와 aria-describedby 속성과 연결된 추가 설명까지 함께 읽는다.

사용자가 입력하기 전에 입력 지침을 알 수 있도록 aria-label=“제목”과 aria-describedby=“입력지침”을 사용하여 아래 그림과 같이 제공한다.

〈input〉 요소에 입력 지침이 포함된 aria-label 속성을 사용한 그림



위와 같이 제작된 아이디/패스워드 입력 페이지를 스크린리더로 들어보면 아래와 같이 들린다.

스크린리더(NVDA) 음성

- 아이디
- 5~20자의 영문 소문자, 숫자와 특수기호 (_), (–)만 사용하여 입력해 주세요. 편집창 아이디
- 패스워드
- 6~16자 영문 대 소문자, 숫자, 특수문자를 사용하여 입력해 주세요. 편집창 패스워드

위와 같이 제작한 후 입력 포맷에 맞지 않아 에러가 발생했을 때는 〈input〉 요소에 aria-describedby 속성에 추가로 에러메시지의 id와 연결해 주고 에러메시지 발생을 알리기 위해 aria-invalid="true"를 추가하고 포커스를 〈input〉 요소로 보내면 에러 발생 상황과 명확한 오류 발생 원인을 스크린리더 사용자에게 알려줄 수 있다.

• WAI-ARIA 속성 정리

아이디와 패스워드 입력상자에 aria-label 속성을 사용하여 제목을 제공하고 aria-describedby 속성을 사용하여 입력 지침을 제공한다.

요소	역할	속성/상태	설명
〈input〉	제목	aria-label="{제목}"	아이디 또는 패스워드와 같은 폼 요소의 제목을 제공
〈input〉	입력지침	aria-describedby="{입력지침}"	폼 요소의 입력 지침을 설정하여 제공한다.

• 소스 코드 및 정리

화면에 보이지 않는 〈label〉 요소를 사용하지 않고 그 역할을 대신 할 수 있는 aria-label 속성을 사용하여 제목을 입력하고 aria-describedby 속성으로 입력지침과 연결하여 아래와 같이 제공한다.

```
<input type="text" aria-label="아이디" aria-describedby="id-helper"
       title="아이디" placeholder="아이디">
<span id="id-helper">
    5~20자의 영문 소문자, 숫자와 특수기호(_), (-)만 사용하여 입력해 주세요.
</span>
<input type="text" aria-label="패스워드" aria-describedby="pw-helper"
       title="패스워드" placeholder="패스워드">
<span id="pw-helper">
    6~16자 영문 대 소문자, 숫자, 특수문자를 사용하여 입력해 주세요.
</span>
```

aria-label 속성과 title 속성이 중복으로 제목을 제공하고 있으므로 title 속성은 삭제해도 무방하지만 국내 접근성 기준 항목에 위배되므로 삽입하였다.

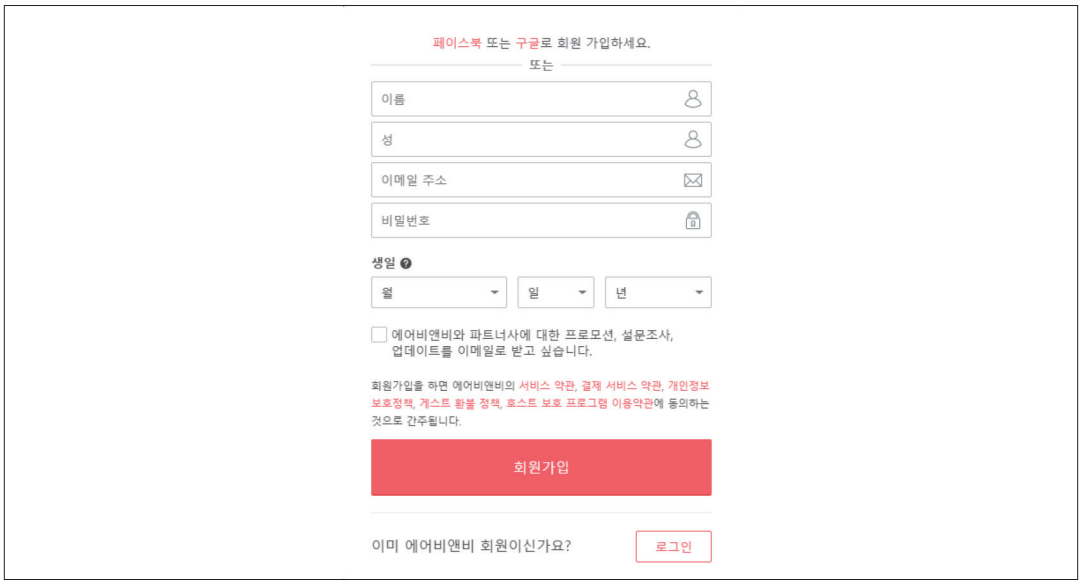
4. 다중 폼(form)서식

• 기존 코드의 문제점들

폼 서식이 있는 UI(User Interface)를 제작할 시 종종 하나의 정보에 대해 여러 개의 폼 관련 요소(form-associated element)들을 사용해야 하는 경우를 만날 수 있다. 이러한 경우를 위해 HTML에서는 〈fieldset〉과 〈legend〉 요소(Element)를 제공하고 있는데, 이것을 이용하면 유저 에이전트는 각각을 그룹핑 콘텐츠와 그 그룹의 제목으로 인식할 수 있다.

만일, 여려 개의 폼 관련 요소들이 하나의 그룹을 이루고 있을 때 이것들이 그룹핑 되어 있지 않을 경우, 사용자는 이 정보를 받아들이는데 있어 어려움을 겪는 상황이 연출 될 수 있다.

airbnb 회원 가입 양식



얼핏 화면만 보면 크게 문제가 될 것이 없어 보이는 가입 양식이다. 이 중 "생일"이라고 굵게 표기 된 부분과 "월", "일", "년" 3개의 셀렉트박스(selectbox)가 하나의 그룹핑 된 정보라는 것에는 이견이 없을 것이다.

아래 해당 부분에 대한 코드를 보도록 하자.

```
<div class="control-group space-top-3 space-1">
    <strong>생일</strong>
    <strong><i class="icon icon-question"></i></strong>
</div>
<div class="control-group row row-condensed space-2">
    <div class="col-sm-5">
        <div class="select va-container-h">
            <select id="user_birthday_month" name="user[birthday_month]">
                <option value="">월</option>
                <option value="1">1월</option>
                  …
            </select>
        </div>
    </div>
    <div class="col-sm-3">
        <div class="select va-container-h">
            <select id="user_birthday_day" name="user[birthday_day]">
```

```
                <option value="">일</option>
                <option value="1">1</option>
                  …
            </select>
        </div>
    </div>
    <div class="col-sm-4">
        <div class="select va-container-h">
            <select id="user_birthday_year" name="user[birthday_year]">
                <option value="">년</option>
                <option value="2016">2016</option>
                  …
            </select>
        </div>
    </div>
</div>
```

이 마크업의 경우 브라우저 자체가 제공하는 〈select〉 요소(Element)를 사용하고는 있으나, 〈fieldset〉과 〈legend〉 요소가 존재하지 않는다.

비 시각장애인에게는 앞선 굵은 글씨의 "생일"이라는 텍스트와 월, 일, 년이 선택되어 있는 3개의 선택 컨트롤을 한 번에 시각 정보로 받기 때문에 하나의 그룹으로 쉽게 이해할 수 있지만, 스크린리더 사용자의 경우, 포커스 모드로만 접근할 경우 월, 일, 년 3개의 선택 컨트롤에 순차적으로 한 번씩 접근이 되기 때문에 이것이 그룹핑 된 정보라는 것 자체를 인식 하기가 어렵다.

생일 부분에 대한 포커스 이동(탭키 이동) 시 스크린리더 음성 출력

NVDA 음성 출력 뷰어

```
이름 편집창
이름
성 편집창
성
이메일 주소 편집창
이메일 주소
비밀번호 편집창 보호됨
비밀번호
콤보박스 월 축소됨
콤보박스 일 축소됨
콤보박스 년 축소됨
에어비앤비와 파트너사에 대한 프로모션, 설문조사, 업데이트를 이메일로 받고 싶습니다.
체크상자 선택됨
```

또한, 위 스크린리더 음성 출력에서 보듯이, "월, 일, 년" 이라는 정보만을 받아들이게 되므로 뒤로 돌아가 가상커서 모드로 변경하여 "생일"이라는 텍스트를 읽기 전까지는 이 컨트롤들이 생일에 관련된 컨트롤임을 인지하기는 더욱 어려운 문제가 있다.

• WAI-ARIA를 사용해야 하는 이유

앞선 예의 문제를 해결하기 위한 가장 손쉬운 방법은 〈fieldset〉과 〈legend〉 요소를 적용하는 방법이다.

```
<fieldset>
    <div class="control-group space-top-3 space-1">
        <legend><strong>생일</strong></legend>
        <strong><i class="icon icon-question"></i></strong>
    </div>
    <div class="control-group row row-condensed space-2">
        <div class="col-sm-5">
            <div class="select va-container-h">
               <select id="user_birthday_month"
                       name="user[birthday_month]">
                    <option value="">월</option>
                    <option value="1">1월</option>
                    …
                </select>
            </div>
        </div>
        <div class="col-sm-3">
            <div class="select va-container-h">
               <select id="user_birthday_day"
                       name="user[birthday_day]">
                    <option value="">일</option>
                    <option value="1">1</option>
                    …
                </select>
            </div>
        </div>
        <div class="col-sm-4">
            <div class="select va-container-h">
               <select id="user_birthday_year"
                       name="user[birthday_year]">
                    <option value="">년</option>
                    <option value="2016">2016</option>
                    …
                </select>
            </div>
        </div>
    </div>
</fieldset>
```

하지만, 기본 컨트롤이 아닌 커스텀 컴포넌트(Custom Component)등을 사용하는 경우와 같이 〈fieldset〉과 〈legend〉 요소(Element)를 사용하지 못하는 경우가 발생 할 수 있는데, 이러한 경우 WAI-ARIA의 group 역할(Role)을 사용하면 〈fieldset〉과 동등한 의미를 전달 할 수 있고, 여기에 aria-labelledby 속성(Property)을 사용함으로 〈legend〉 요소(Element)가 가지는 의미 또한 전달이 가능해 진다.

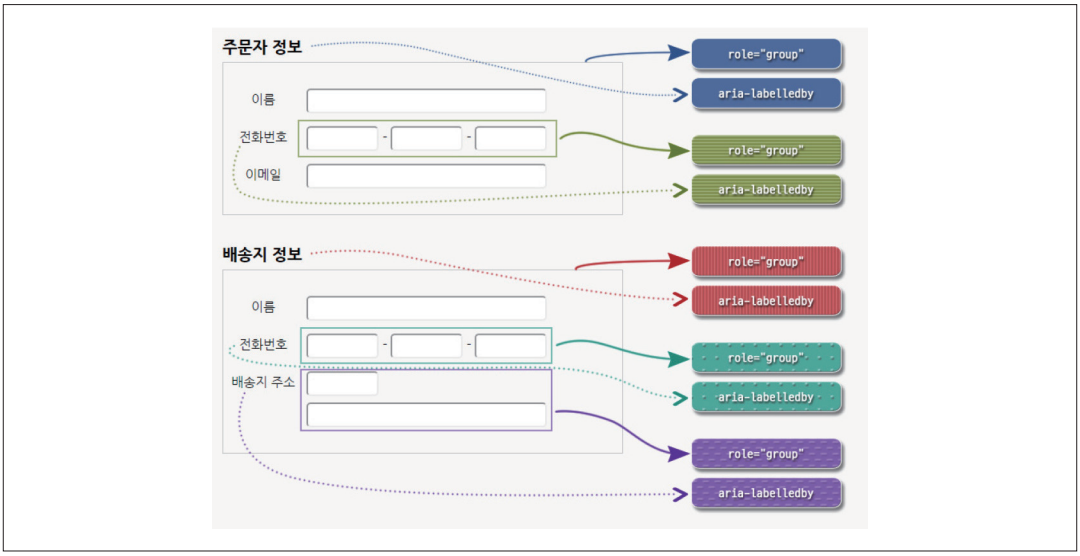
물론, 당연히 HTML의 기본 요소(〈fieldset〉과 〈legend〉)를 사용하는 것이 항상 우선이다. WAI-ARIA의 적용은 이것이 불가할 경우에 대한 차선책으로 선택할 수 있는 보조 기술이다.

• WAI-ARIA 전체적인 그림과 설명

먼저, WAI-ARIA의 전체적인 역할(Role)과 속성(Property) 구조를 그림으로 살펴보자.

예제에서는 비교를 좀 더 부각하기 위해 별도의 폼을 구성해 두었다.

WAI-ARIA 역할(Role), 속성(Property) 구조



그룹이라는 정보를 제공하기 위해 해당 컨트롤들을 감싸는 요소에 role="group"을 사용하고, 이 요소에 그룹의 레이블을 설정하기 위해 aria-labelledby 속성으로 레이블 정보를 가지는 요소의 id 값을 연결해주면 스크린리더는 해당 요소로부터 특정한 이름을 가진 그룹으로 정보를 제공할 수 있다.

다음에 등장하는 그림은 WAI-ARIA를 적용한 서식을 스크린리더에서 어떻게 읽어주는지를 나타내고 있다.

이때 스크린리더 테스트는 탭 이동을 통해 폼 관련 요소(form-associated element)들을 탐색하는 경우에 대해 진행하였다.

WAI-ARIA 적용에 따른 스크린리더 음성 출력

WAI-ARIA 적용 이전	WAI-ARIA 적용 이전
NVDA 음성 출력 뷰어 이름 편집창 빈줄 국번 편집창 빈줄 가운데 자리 편집창 빈줄 마지막 자리 편집창 빈줄 이메일 편집창 빈줄 이름 편집창 빈줄 국번 편집창 빈줄 가운데 자리 편집창 빈줄 마지막 자리 편집창 빈줄 우편번호 편집창 빈줄 상세 주소 편집창 빈줄	NVDA 음성 출력 뷰어 주문자 정보 그루핑 이름 편집창 빈줄 전화번호 그루핑 국번 편집창 빈줄 가운데 자리 편집창 빈줄 마지막 자리 편집창 빈줄 이메일 편집창 빈줄 배송지 정보 그루핑 이름 편집창 빈줄 전화번호 그루핑 국번 편집창 빈줄 가운데 자리 편집창 빈줄 마지막 자리 편집창 빈줄 배송지 주소 그루핑 우편번호 편집창 빈줄 상세 주소 편집창

스크린리더 음성 출력 결과에서 보이듯, WAI-ARIA가 적용된 이후에는 각 폼 관련 요소(form-associated element)들을 탭키를 이용하여 탐색하는 경우에도 사용자가 접근한 요소가 어떤 그룹에 속하는 정보에 관련된 것인지를 명확하게 알 수 있어 정보를 인지하는 것이 매우 수월하다.

• WAI-ARIA 속성 정리

각 요소에 사용할 역할(Role)과 속성(Property)을 정리해보면 다음과 같다.

요소	역할	속성/상태	설명
div	group		폼에 공통적인 레이블을 참조할 수 있는 필드의 논리적 집합을 그룹핑
		aria-labelledby	그룹의 레이블에 해당하는 요소의 ID를 설정

• 소스 코드 정리

이제, 앞서 설명한 대로 WAI-ARIA를 적용시켜 보도록 하자.

적용을 위해서 그룹으로 묶는 요소에 role="group"을 부여하고, 이 그룹에 레이블을 명시하기 위해 각 그룹의 제목에 해당하는 요소의 id 값을 aria-labelledby 속성의 값을 설정하면 된다.

```
<span id="order-info" class="form-legend">주문자 정보</span>
<div class="field" role="group" aria-labelledby="order-info">
    <div class="field-item">
        <label for="usr-name" class="label">이름</label>
        <input type="text" name="usr-name" id="usr-name" />
    </div>
    <div class="field-item" role="group"  aria-labelledby="label-tel">
        <span class="label" id="label-tel">전화번호</span>
        <input type="text" name="usr-tel-1" id="usr-tel-1"
               aria-label="국번" /> -
        <input type="text" name="usr-tel-2" id="usr-tel-2"
               aria-label="가운데 자리"  /> -
        <input type="text" name="usr-tel-3" id="usr-tel-3"
               aria-label="마지막 자리"  />
    </div>
    <div class="field-item">
        <label for="usr-email">이메일</label>
        <input type="text" name="usr-email" id="usr-email" />
    </div>
</div>
<span id="delivery-info" class="form-legend">배송지 정보</span>
<div class="field" role="group" role="grouaaria-labelledby="delivery-info"p">
    <div class="field-item">
        <label for="deliver-name" class="label">이름</label>
        <input type="text" name="deliver-name" id="deliver-name" />
    </div>
    <div class="field-item" role="group" aria-labelledby="label-tel2">
                <span class="label" id="label-tel2">전화번호</span>
                 <input type="text" name="deliver-tel-1" id="deliver-tel-1"
               aria-label="국번" /> -
        <input type="text" name="deliver-tel-2" id="deliver-tel-2"
              aria-label="가운데 자리"  /> -
         <input type="text" name="deliver-tel-3" id="deliver-tel-3"
```

```
                aria-label="마지막 자리"  />
    </div>
    <div class="field-item" role="group" aria-labelledby="label-addr">
        <span class="label" id="label-addr">배송지 주소</span>
        <input type="text" name="deliver-post" id="deliver-post"
               aria-label="우편번호" />
        <input type="text" name="deliver-addr" id="deliver-addr"
               aria-label="상세 주소" />
    </div>
</div>
```

5. 버튼(button)

• 기존 코드의 문제점들

버튼은 여러 가지 기능을 수행을 하는 중요한 컴포넌트 중 하나이다. 폼 서식을 제출하기 위한 버튼, 레이어 팝업을 열기 위한 버튼, 상태 전환을 위한 토글 버튼, 취소나 삭제를 수행하는 위한 버튼 등 수많은 종류의 버튼이 있다.

버튼과 링크는 엄연히 다른 기능이다. 새로운 리소스를 가져오거나 새로운 페이지를 불러오기 위한 것은 링크이기때문에 〈a〉 요소로 사용해야 하고, 데이터의 업데이트나 변경, 레이어 팝업 열기, 페이지의 특정 요소를 열거나 닫는 기능으로 어떤 특정 동작을 하는 것은 버튼의 역할이므로 〈button〉 요소로 제작되어야 한다.

WAI-ARIA 적용에 따른 스크린리더 음성 출력

이러한 구분 없이 버튼 기능을 〈a〉 요소로 제작하거나 키보드가 접근이 되지 않는 〈span〉이나 〈div〉 등의 요소로 제작하는 경우가 많다.

〈a〉 요소의 href 속성은 hypertext reference의 약자로 이동할 페이지의 주소를 적어주어야 한다. 그러나 이동할 페이지의 주소 대신 자바스크립트 함수를 실행하기 위해 슈도 프로토콜(Pseudo Protocol)인 javascript:함수명 형식을 사용하는 것은 〈a〉 요소의 사용 목적과도 맞지 않으며 올바르게 사용했다고 볼 수 없다.

```
<a href="javascript:alert('hello');">hello</a>
```

또한 〈div〉 또는 〈span〉 요소를 버튼으로 제작한 경우, 마우스로는 기능을 수행할 수 있지만 키보드 사용자나 스크린리더 사용자는 탭키 또는 가상모드 탐색 시 버튼을 인지하거나 사용할 수 없어 접근성에 치명적인 문제를 발생시킨다.

```
<div class="button" id="print">Print Page</div>
```

아래 그림은 〈div〉 요소를 버튼으로 사용한 그림이다. 〈div〉 요소로 버튼을 제작했으므로 클릭이 가능한지 인지가 쉽지 않고, 마우스 포인터로 사용해도 버튼인지 시각적으로 구분되지 않는다. 스크린리더에서는 "Print Page"로만 읽기 때문에 버튼이라는 것을 전혀 인지할 수 없다.

〈div〉 요소를 버튼으로 사용한 경우


```
<div>Print<br>Page<div>
```

스크린리더(NVDA) 음성

Print Page

또한 〈a〉 요소를 사용하여 버튼으로 제작한 경우, 스크린리더 사용자는 버튼이 아닌 링크로 인식하게 되어 사용하는 데 혼돈을 느끼게 된다. 또한 〈a〉 요소를 버튼으로 사용할 때 href 속성을 삭제하는 경우가 있는데 그렇게 되면 키보드의 포커스를 받지 못하므로 삭제하지 않아야 한다.

• WAI-ARIA를 사용해야 하는 이유

〈a〉 요소 또는 〈div〉, 〈span〉 요소를 버튼으로 사용한 경우 WAI-ARIA를 사용하면 마우스 사용자만이 아니라 키보드 사용자와 스크린리더 사용자들도 버튼 기능을 명확히 인지하여 운용할 수 있다.

그럼, WAI-ARIA를 사용하여 〈a〉 요소 또는 〈div〉나 〈span〉 요소로 마크업한 버튼의 접근성을 개선하는 방법을 알아보자.

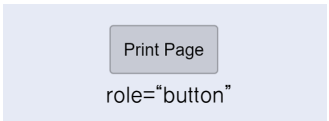
버튼 역할을 하는 요소에 role="button" 을 지정하면 스크린리더는 해당 요소를 버튼으로 인식하여 읽는다. 또한 토글 버튼의 경우 현재 상태를 알려주어야 하는데 aria-pressed 속성을 사용하면 현재 상태가 눌려져 있는지(true), 눌려져 있지 않은 상태인지 (false)를 스크린리더가 인지하여 읽을 수 있다.

• WAI-ARIA 전체적인 그림과 설명

〈button〉 또는 〈input〉 요소 이외의 요소로 버튼을 제작한 경우는 버튼의 역할을 명시해 주어야 한다.

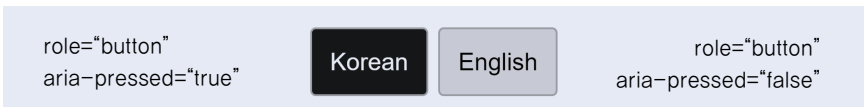
버튼의 역할을 부여하기 위해 role="button"을 삽입하고, 엔터키와 스페이스바로 선택이 가능하도록 이벤트를 걸어야 한다.

버튼으로 사용하기 위해 role 속성을 부여한 경우



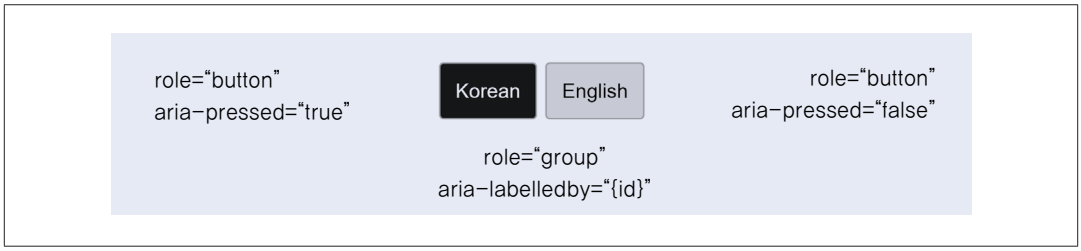
토글 버튼도 각각 role="button"을 삽입하고, 현재 눌려져 있는 상태인지를 aria-pressed 속성을 사용하여 정보를 제공할 수 있다.

토글 버튼으로 사용하기 위해 button 역할과 aria-pressed 속성을 지정한 경우



2개 이상의 토글 버튼은 그룹화 할 수 있으며, role="group"을 버튼을 감싸고 있는 컨테이너에 삽입하고 aria-labelledby 속성을 사용하여 값은 제목이 포함된 요소의 id 값으로 지정한다.

토글 버튼의 그룹화



• WAI-ARIA 속성 정리

role="button"은 해당 요소의 역할을 버튼으로 정의하고, 선택적으로 aria-describedby 속성을 사용하여 버튼에 추가 설명을 삽입할 수 있다. 버튼이 비 활성화 되었거나 더 이상 사용할 수 없을 때 aria-disabled 속성을 사용하여 현재의 상태 정보를 스크린리더에게 알려줄 수 있다.

토글 버튼은 눌려져 있거나 눌려져 있지 않은 두 가지 경우 중 하나이므로, 현재 버튼의 상태 정보를 aria-pressed 속성을 사용하여 알려줄 수 있다. 또한 2개 이상의 토글 버튼을 그룹화하여 제공하면 스크린리더가 사용자의 쉬운 이해를 도울 수 있다.

단, 토글 버튼은 눌려지거나 눌려지지 않는 상태이므로 aria-disabled 속성을 사용하지 않으며 눌려지는 상태 변화에 따라 버튼 텍스트도 변경되어서는 안 된다.

요소	역할	속성과 상태	설명
〈button〉	button		버튼 역할을 정의하며, 엔터키나 스페이스바로 선택
		aria-disabled ="true | false"	사용할 수 없는 버튼 상태는 true, 사용할 수 있는 상태는 false를 사용
〈button〉 토글	button		버튼 역할을 정의하며, 엔터키나 스페이스바로 선택
		aria-pressed ="true | false"	버튼의 상태가 눌려져 있으면 true, 눌려져 있지 않으면 false를 사용
	group		2개 이상의 토글 버튼을 그룹화
		aria-labelledby=""	그룹의 제목을 제공하기 위한 것으로 제목 요소에 삽입된 id와 연결

• 소스 코드 정리

다음은 〈div〉 요소로 접근 가능한 버튼을 제작하는 코드 사례이다.

```
$(document).ready(function(){
    $("#div-btn").click(function(){
       alert("이것은 버튼입니다.");
    });
});
<div class="button" id="div-btn">DIV버튼</div>
```

tabindex 속성을 사용하여 탭키로 이동이 가능하게 하고, WAI-ARIA의 role="button"을 삽입하여 스크린리더에서 버튼임을 인식하게 한다. 이때 스크린리더로 듣게 되면, "button, print page"로 읽어주기 때문에 페이지를 인쇄하기 위한 버튼임을 인지할 수 있다.

```
<div class="button" id="div-btn" role="button" tabindex="0">DIV 버튼</div>
```

role="button"을 적용하고 스크린리더로 들어보면 아래와 같이 들린다.

스크린리더(NVDA) 음성

Button DIV버튼

탭키로 이동하고 엔터키나 스페이스바로 기능이 동작하도록 자바스크립트도 아래와 같이 변경한다.

```
$(document).ready(function() {
    $("#div-btn").click(function() {
        alert("이것은 버튼입니다.");
    });
    $('#div-btn').keyup(function(e) {
        if (e.keyCode == 32) {
            alert('스페이스바를 눌렀군요!');
        }
        if (e.keyCode == 13) {
            alert('엔터키를 눌렀군요!');
        }
    });
});
```

토글 버튼인 경우, WAI-ARIA의 aria-pressed 속성을 사용하여 현재 상태를 스크린리더에게 알려주어야 하는데 눌려졌을 때는 true, 눌려지지 않았을 때는 false로 동적으로 변경되어야 한다.

```
<div class="button" id="language"
     tabindex="0" role="button" aria-pressed="true">
     Korean
</div>
```

aria-pressed 속성을 버튼에 사용하면 토글로 인식하고 스크린리더에서는 아래와 같이 들린다.

스크린리더(NVDA) 음성

Toggle button pressed Korean

더 응용하자면, 2개 이상의 토글 버튼은 그룹화를 하여 스크린리더 사용자가 더 쉽고 빠르게 버튼 기능을 이해할 수 있고 사용할 수 있도록 할 수 있다.

```
<h2 id="language">언어선택</h2>
<div role="group" aria-labelledby="language">
    <div class="tgbutton" id="div-btn1" role="button" tabindex="0"
         aria-pressed="true">Korean</div>
    <div class="tgbutton" id="div-btn2" role="button" tabindex="0"
         aria-pressed="false">English</div>
 </div>
```

• 키보드 인터랙션

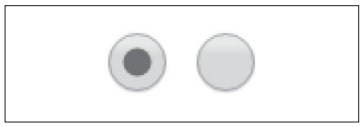
키보드	인터랙션
엔터키	버튼의 선택, 토글버튼의 선택 해제
스페이스바	버튼의 선택, 토글버튼의 선택 해제

6. 라디오버튼(Radio Button)

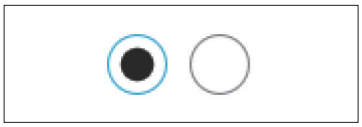
• 기존 코드의 문제점들

일반적으로 라디오버튼은 브라우저마다 디자인이 조금씩 상이하기 때문에 디자이너들은 일관된 디자인 표현을 위해서 커스텀 컴포넌트를 사용하는 경우가 많다.

Chrome의 기본 라디오버튼



IE의 기본 라디오버튼



라디오버튼의 디자인 변경을 위해서 〈input type="radio"〉 요소는 CSS의 opacity 속성으로 화면에서 보이지 않도록 처리한후 배경이미지를 활용하여 원하는 라디오버튼의 디자인을 적용하는 방법이 있고, 또 하나는 〈input〉 요소을 사용하지 않고 〈span〉이나 〈div〉 요소, 또는 〈ul〉, 〈li〉 요소를 사용하여 마크업하고 이를 CSS로 디자인 하는 경우를 들 수 있다.

이렇게 제작된 코드는 기본 라디오버튼과 동일하게 키보드 인터랙션이 보장되어야 한다.

예를 들면, 라디오 버튼임을 인식할 수 있어야 하고, 선택과 해제된 상태를 구분할 수 있어야 한다. 또한 키보드의 스페이스바로 선택이 가능해야 하지만 이런 부분을 간과하고 제작하는 경우가 많다.

• WAI-ARIA를 사용해야 하는 이유

WAI-ARIA를 사용하면 키보드 사용 보장도 가능하고 스크린리더 사용자도 사용할 수 있는 디자인된 라디오버튼을 사용할 수 있다.

라디오버튼은 일반적으로 YES or NO와 같이 두 개 이상이 항상 존재한다.

라디오버튼을 role="radiogroup" 또는 role="group"을 사용하여 그룹화하고 aria-label 또는 aria-labelledby 속성을 사용하여 그룹의 제목을 스크린리더 사용자들에게 알려줄 수 있다. 이때 radio 역할이 스크린리더 사용자에게 라디오버튼임을 인식할 수 있도록 해준다.

또한, 스크린리더가 라디오버튼이 현재 선택되어 있는지, 선택되지 않았는지의 상태정보를 인식할 수 있게 aria-checked 속성을 사용할 수 있다.

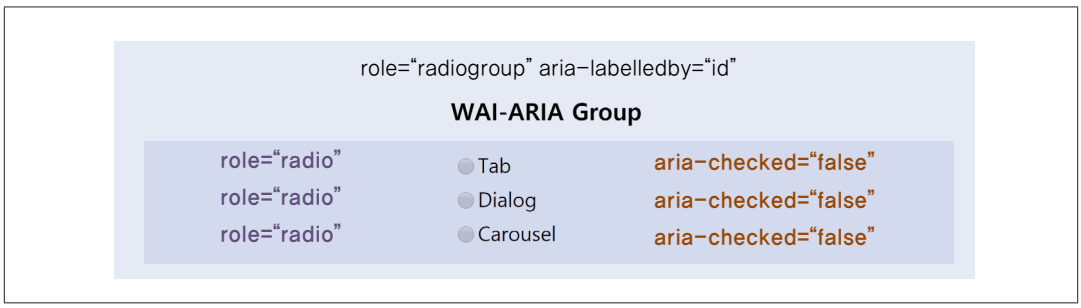
• WAI-ARIA 전체적인 그림과 설명

라디오버튼 그룹을 마크업할 때는 스크린리더에서 전체의 목록 개수를 읽어줄 수 있도록 〈ul〉, 〈li〉 요소로 제작하는 것을 권장한다.

전체를 감싸고 있는 컨테이너인 〈ul〉 요소에 role="radiogroup"을 지정하고. 각 〈li〉 요소에는 각각 role="radio"를 삽입 하여 라디오 버튼의 역할을 정의한다. 단, role="radiogroup"은 하위 요소가 role="radio"일 경우에 사용하며, 〈input type="radio"〉 요소를 라디오 그룹으로 묶을 경우는 role="group"을 사용한다.

전체적인 그림을 보면 아래 그림과 같다.

〈ul〉, 〈li〉 요소를 라디오 버튼으로 제작한 사례



키보드 방향키로 라디오버튼을 선택하면서 스크린리더로 들어보면 아래와 같이 들린다.

스크린리더(NVDA) 음성

- WAI-ARIA Group grouping
- Tab radio button checked 1 of 3
- dialog radio button checked 2 of 3
- carousel radio button checked 3 of 3

• WAI-ARIA 속성정리

role="radiogroup"은 그룹이므로 그룹의 제목을 함께 제공해야 한다. 주로 제목으로 마크업되어 있는 헤딩 요소에 id 값을 지정하여, aria-labelledby 속성으로 연결한다. 이때 스크린리더로 들어보면 헤딩 텍스트를 그룹의 제목으로 인식하는 것을 알수 있다.

role="radio"가 지정된 〈li〉 요소에는 aria-checked 속성을 사용하여, 선택된 라디오버튼일 경우 true, 선택되지 않은 라디오버튼일 경우 false로 지정하면 스크린리더가 라디오버튼의 상태 정보를 읽을 수 있다.

또한 tabindex 속성을 사용하여 키보드 포커스를 받을 수 있도록 할 수 있다.

요소	역할	속성/상태	설명
〈ul〉	radiogroup		라디오버튼 그룹의 역할 정의
		aria-labelledby=""	라디오버튼 그룹의 제목으로 주로 헤딩의 id와 연결. (연결할 수 있는 제목이 없을 경우 aria-label 속성을 사용하여 제목 제공)
〈li〉	radio		라디오버튼 역할 정의
		aria-checked ="true | false"	라디오버튼이 선택되어 있는지, 선택되지 않았는지 상태 정보 제공

• 소스 코드 정리

다음은 〈ul〉, 〈li〉 요소로 라디오 버튼을 제작하는 코드 사례이다. 가장 먼저 제목을 마크업 하고 해당 요소에 id를 지정한다.

```
<h1 id="rg1_label">WAI-ARIA Group</h1>
```

라디오버튼의 그룹과 라디오 버튼으로 사용할 〈ul〉, 〈li〉 요소에도 각각 id를 지정하고 〈li〉 요소에 키보드 포커스를 받을 수 있도록 하기 위해 tabindex="0"을 추가로 지정한다.

```
<ul class="radiogroup" id="rg1">
    <li id="r1" tabindex="0">tab 1</li>
    <li id="r2" tabindex="0">dialog 2</li>
    <li id="r3" tabindex="0">carousel 3</li>
</ul>
```

radiogroup 역할을 〈ul〉 요소에 지정하고 〈h1〉 요소의 id 값을 aria-labelledby 속성에 지정한다.

```
<ul role="radiogroup" aria-labelledby="rg1_label" class="radiogroup"
    id="rg1">
```

〈li〉 요소에는 role="radio"를 추가로 지정하고 aria-checked 속성을 사용하여 상태정보를 인식할 수 있도록 한다. 이때 라디오버튼이 선택된 상태라면 "true" 값을 선택되지 않은 상태라면 "false" 값을 지정한다.

첫번째 라디오버튼을 선택했을 때의 전체 마크업 소스는 아래와 같다.

```
<h1 id="rg1_label">WAI-ARIA Group</h1>
<ul aria-labelledby="rg1_label" class="radiogroup" id="rg1"
    role="radiogroup">
    <li aria-checked="true" id="r1" role="radio" tabindex="0">tab 1</li>
    <li aria-checked="false" id="r2" role="radio" tabindex="-1">
        dialog 2
    </li>
    <li aria-checked="false" id="r3" role="radio" tabindex="-1">
        carousel 3
    </li>
</ul>
```

자바스크립트로 제작을 할 때는 아래 소스를 참고하여 제작이 가능하다.

role="radio"가 지정된 〈li〉 요소와 aria-checked="true"로 지정된 값을 함수로 저장한다.

```
// role="radio"를 갖고 있는 <li>요소와 aria-checked="true"인 것을 참조
function radioGroup(id) {
     this.$buttons = this.$id.find('li').filter('[role=radio]');
     this.$checked = this.$buttons.filter('[aria-checked=true]');
}
```

```
radioGroup.prototype.selectButton = function($id) {
    // 버튼이 선택이 되면 aria-checked="true"가 되고, tabindex="0"을 삽입
    // 선택되지 않은 라디오버튼은 tabindex="-1"로 변경
    if (this.checkButton == true) {
        this.$checked.attr('aria-checked', 'false');
        $id.attr('aria-checked', 'true');
        if (this.$checked.length == 0) {
            this.$buttons.first().attr('tabindex', '-1');
            this.$buttons.last().attr('tabindex', '-1');
        } else {
            this.$checked.attr('tabindex', '-1');
        }
        $id.attr('tabindex', '0');
        this.$checked = $id;
    }
```

키보드 인터랙션을 위해 아래와 같이 KeyDown과 KeyUp의 이벤트핸들러를 적용한다.

```
radioGroup.prototype.handleKeyDown = function(e, $id) {
    if (e.altKey) {
        return true;
    }
    // 위쪽 방향키와 왼쪽 방향키를 누르면 이전 라디오버튼으로 이동하고
    // 처음 라디오 버튼일 경우 마지막으로 이동한다.
    switch (e.keyCode) {
        case this.keys.left:
        case this.keys.up:
            {

                var $prev = $id.prev();
                if (e.shiftKey) {
                    return true;
                }
                if ($id.index() == 0) {
                    $prev = this.$buttons.last();
                }
                if (e.ctrlKey) {
                    this.checkButton = false;
                }
                $prev[0].focus();
                e.stopPropagation();
                return false;
            }
        // 아래쪽 방향키와 오른쪽 방향키를 누르면 다음 라디오버튼으로 이동하고
        // 마지막 라디오 버튼일 경우 처음으로 이동한다.
        case this.keys.right:
        case this.keys.down:
            {
                var $next = $id.next();
                if (e.shiftKey) {
                    return true;
                }
                if ($id.index() == this.$buttons.length - 1) {
                    $next = this.$buttons.first();
                }
                if (e.ctrlKey) {
                    this.checkButton = false;
                }
                $next[0].focus();
                e.stopPropagation();
                return false;
            }
    }
    return true;
};
```

• 키보드 인터랙션

키보드	인터랙션
탭키	라디오버튼 그룹으로의 포커스 이동
위/왼쪽 방향키	라디오버튼 그룹 내에서 이전 라디오버튼으로 포커스를 이동시키고 선택. (단, 첫번째 라디오버튼일 경우 마지막 라디오버튼으로 이동)
아래/오른쪽 방향키	라디오버튼 그룹 내에서 이전 라디오버튼으로 포커스를 이동시키고 선택. (단, 첫번째 라디오버튼일 경우 마지막 라디오버튼으로 이동)
스페이스바	(선택된 기본 값이 없을 때) 라디오버튼 선택

7. 체크박스(Checkbox)

• 기존 코드의 문제점들

체크박스도 라디오버튼과 같이 브라우저마다 디자인이 조금씩 상이하여 커스터마이징된 디자인을 적용하는 경우가 많다.

커스터마이징 디자인이 적용된 다양한 체크박스

이미 온라인 상으로 많이 배포되어 있는 무료 jQuery 플러그인 소스를 보면 〈input type="checkbox"〉를 CSS로 숨기고 디자인을 적용하거나 〈ul〉, 〈li〉 요소를 사용하는 경우가 있다.

대개 이렇게 제작된 많은 경우, 마우스로만 사용 가능하도록 제작 되고 키보드로는 동작이 안되거나 스크린리더가 체크박스임을 인지하지 못하는 경우가 많다.

• WAI-ARIA를 사용해야 하는 이유

라디오버튼과 같이 체크박스도 WAI-ARIA 역할인 role="checkbox"가 있다. 이 role을 사용하면 스크린리더는 기존의 〈li〉 요소를 체크박스와 같이 인식하게 되고 aria-checked 속성 값을 인지하여 체크 여부를 읽어주게 되어 스크린리더 사용자들도 일반인들과 동일하게 체크박스를 사용할 수 있다.

또한 브라우저에서 제공하는 기본 체크박스는 스페이스바로 선택이 가능하므로, 자바스크립트 이벤트 핸들러로 스페이스바의 동작을 마우스 클릭과 동일하게 제공해야 한다

• WAI-ARIA 전체적인 그림과 설명

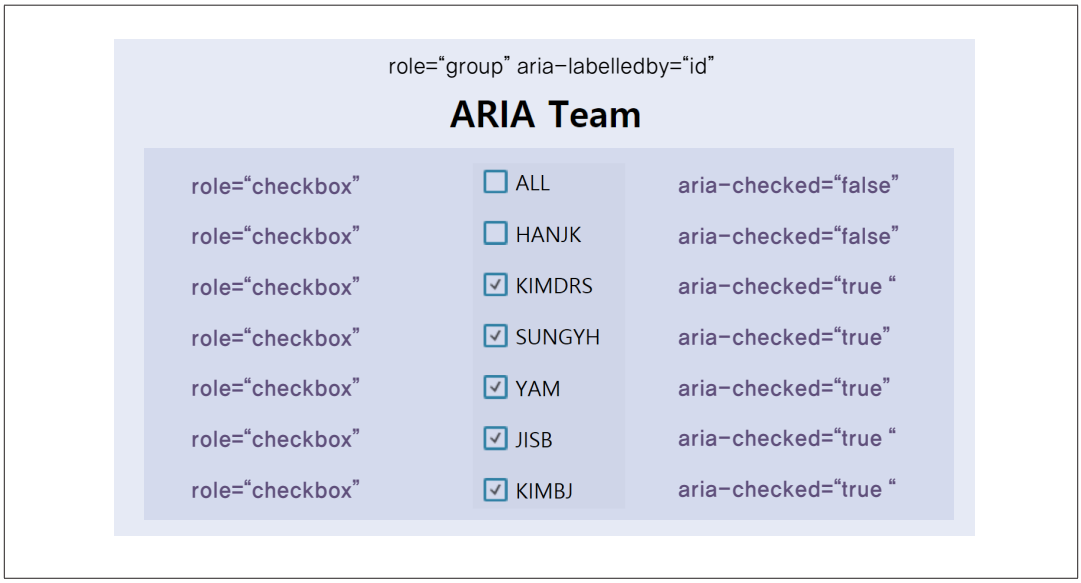
체크박스의 전체 그룹을 〈ul〉 요소로 마크업하고 바로 상위에 헤딩 요소를 사용한다.

〈ul〉 요소에 role="group"을 사용하여 그룹화하고, 상위 헤딩 요소에 삽입된 id 값과 aria-labelledby 속성을 연결하여 그룹의 제목을 지정해 준다.

〈li〉 요소는 각 role="checkbox"를 이용하여 체크박스 역할을 지정해주고 aria-checked 속성의 true와 false 값을 삽입하여 체크박스의 체크 상태 정보를 제공한다.

전체적인 그림을 보면 아래 그림과 같다.

〈ul〉, 〈li〉 요소로 체크박스 그룹을 제작한 이미지



탭키로 체크박스를 이동하면서 스크린리더로 들어보면 아래와 같이 들리고, 스페이스바를 누르면 체크박스 선택 속성이 바뀌면서 스크린리더 음성은 “not checked”가 “checked”로 변경된다.

스크린리더(NVDA) 음성

- all check box not checked
- HANJK check box not checked
- KIMDRS check box checked
- SUNGYH check box checked
- YAM check box checked
- JISB check box checked
- KIMBJ check box checked

• WAI-ARIA 속성 정리

전체 체크박스 그룹을 role=“group”으로 정의하고 그룹의 제목을 제목 콘텐츠의 id와 aria-labelledby 속성를 연결하거나 aria-label 속성으로 제공한다.

〈li〉 요소에 role=“checkbox”로 정의하고 aria-checked 속성을 사용하여, 선택된 체크박스는 true, 선택되지 않은 체크박스는 false 로 적용하면 스크린리더가 체크박스의 상태 정보를 읽을 수 있다.

포커서블하지 않은 〈li〉 요소를 체크박스로 사용하였으므로 tabindex 속성을 사용하여 키보드 포커스를 받을 수 있도록 할 수 있다.

요소	역할	속성/상태	설명
〈ul〉	group		체크박스 그룹의 역할 정의
		aria-labelledby=“”	체크박스 그룹의 제목으로 주로 헤딩의 id와 연결. (연결할 수 있는 제목이 없을 경우 aria-label 속성을 사용하여 제목 제공)
〈li〉	checkbox		체크박스 역할 정의
		aria-checked =“true | false”	체크박스가 선택되어 있는지, 선택되지 않았는지 상태 정보 제공

• 소스 코드 정리

다음은 〈h2〉, 〈ul〉, 〈li〉 요소로 접근성 있는 체크박스를 제작하는 코드 사례이다.

```
<h2>ARIA Team</h2>
<ul class="checkboxes">
    <li class="checkbox" id="allCheck" tabindex="0"><span>all</span></li>
    <li class="checkbox" id="cb1a" tabindex="0"><span>HANJK</span></li>
    <li class="checkbox" id="cb1b" tabindex="0"><span>KIMDRS</span></li>
    <li class="checkbox" id="cb1c" tabindex="0"><span>SUNGYH</span></li>
    <li class="checkbox" id="cb1d" tabindex="0"><span>YAM</span></li>
    <li class="checkbox" id="cb1d" tabindex="0"><span>JISB</span></li>
    <li class="checkbox" id="cb1d" tabindex="0"><span>KIMBJ</span></li>
</ul>
```

체크박스의 역할을 하고 있는 〈li〉 요소에 role="checkbox"로 삽입하고 현재 체크상태를 aria-checked 속성으로 제공한다.

전체 〈ul〉 요소에 role="group"을 사용하여 그룹으로 묶고 aria-labelledby 속성과 〈h2〉 요소의 id 속성 값을 연결하여 그룹 제목으로 제공한다.

```
<h2 id="h1aria">ARIA Team</h2>
<ul aria-labelledby="h1aria" class="checkboxes" role="group">
    <li aria-checked="false" class="checkbox"
        id="allCheck" role="checkbox" tabindex="0"><span>all</span></li>
    <li aria-checked="false" class="checkbox"
        id="cb1a" role="checkbox" tabindex="0"><span>HANJK</span></li>
    <li aria-checked="true" class="checkbox"
        id="cb1b" role="checkbox" tabindex="0"><span>KIMDRS</span></li>
    <li aria-checked="true" class="checkbox"
        id="cb1c" role="checkbox" tabindex="0"><span>SUNGYH</span></li>
    <li aria-checked="true" class="checkbox"
        id="cb1d" role="checkbox" tabindex="0"><span>YAM</span></li>
    <li aria-checked="true" class="checkbox"
        id="cb1d" role="checkbox" tabindex="0"><span>JISB</span></li>
    <li aria-checked="true" class="checkbox"
        id="cb1d" role="checkbox" tabindex="0"><span>KIMBJ</span></li>
</ul>
```

jQuery를 이용하여 해당 상태 속성이 변경되도록 아래와 같이 제작한다.

```
$(document).ready(function() {
    // checkbox 속성을 갖고 있는 <li> 요소를 클릭할 때
    $('li[role=checkbox]').click(function() {
        // aria-checked 속성이 true이면 false로 변경
        if ($(this).attr('aria-checked') === 'true') {
            $(this).attr('aria-checked', 'false');
        } else {// 그렇지 않으면 true로 변경
            $(this).attr('aria-checked', 'true');
        }
    });
    // id가 #allCheck인 요소를 클릭하면
    $('#allCheck').click(function() {
        // 한개라도 aria-checked 속성과 true 값을 갖고 있다면
        if ($(this).attr('aria-checked') === 'true') {
            // checkbox 속성을 갖고 있는 <li> 요소의 aria-checked 속성은 true로 변경
            $('li[role=checkbox]').attr('aria-checked', 'true');
        } else {
            // 그렇지 않으면 checkbox 속성을 갖고 있는 <li> 요소의
            // aria-checked 속성은 false로 변경
            $('li[role=checkbox]').attr('aria-checked', 'false');
        }
    });
```

동일하게 스페이스바로 동작하도록 아래와 같이 추가 작성한다.

```
// 위와 동일한 이벤트 핸들러이며 스페이스바를 동작을 위해 추가
    $('li[role=checkbox]').keydown(function(e) {
        // 스페이스바 키 번호 32
        if (e.keyCode == '32') {
            if ($(this).attr('aria-checked') === 'true') {
                $(this).attr('aria-checked', 'false');
            } else {
                $(this).attr('aria-checked', 'true');
            }
        }
    });
    // allcheck인 경우도 동일하게 스페이스바가 동작되도록 추가
    $('#allCheck').keydown(function(e) {
```

```
        if (e.keyCode == '32') {
            if ($(this).attr('aria-checked') === 'true') {
                $('li[role=checkbox]').attr('aria-checked', 'true');
            } else {
                $('li[role=checkbox]').attr('aria-checked', 'false');
            }
        }
    });
});
```

• 키보드 인터랙션

키보드	인터랙션
탭키	체크박스 영역으로 키보드 포커스 이동
스페이스바	체크박스의 체크 상태 전환

8. 에러메시지(Error Message)

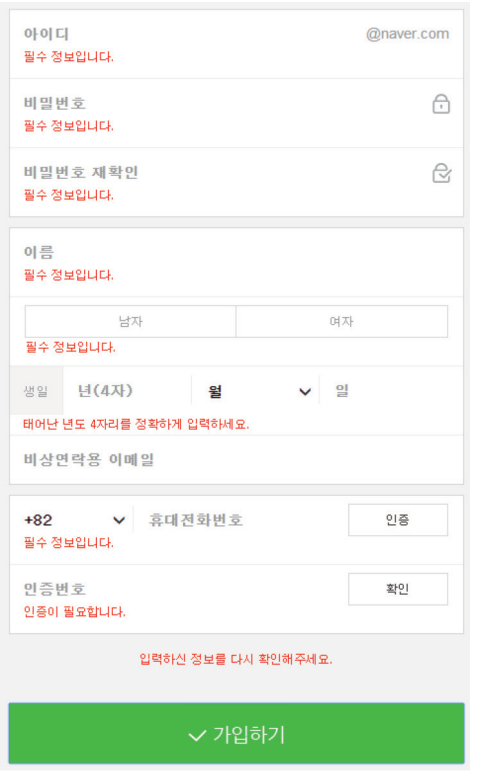
• 기존 코드의 문제점들

시각장애인들은 회원가입과 같이 다양한 입력 포맷을 사용하고 다양한 에러메시지를 인지하는데 어려움이 발생한다

대표적인 포털 사이트인 NAVER와 DAUM의 회원가입 페이지에서 에러가 발생하면 아래와 그림과 같이 보인다. 시력을 갖고 있는 비 스크린리더 사용자라면 에러가 어디에서 발생했고 어떤 에러인지 쉽게 이해가 가능하다.

그러나, 스크린리더 사용자에게 어떻게 들리는지 확인해 보자.

에러가 발생한 NAVER의 회원가입 입력 폼

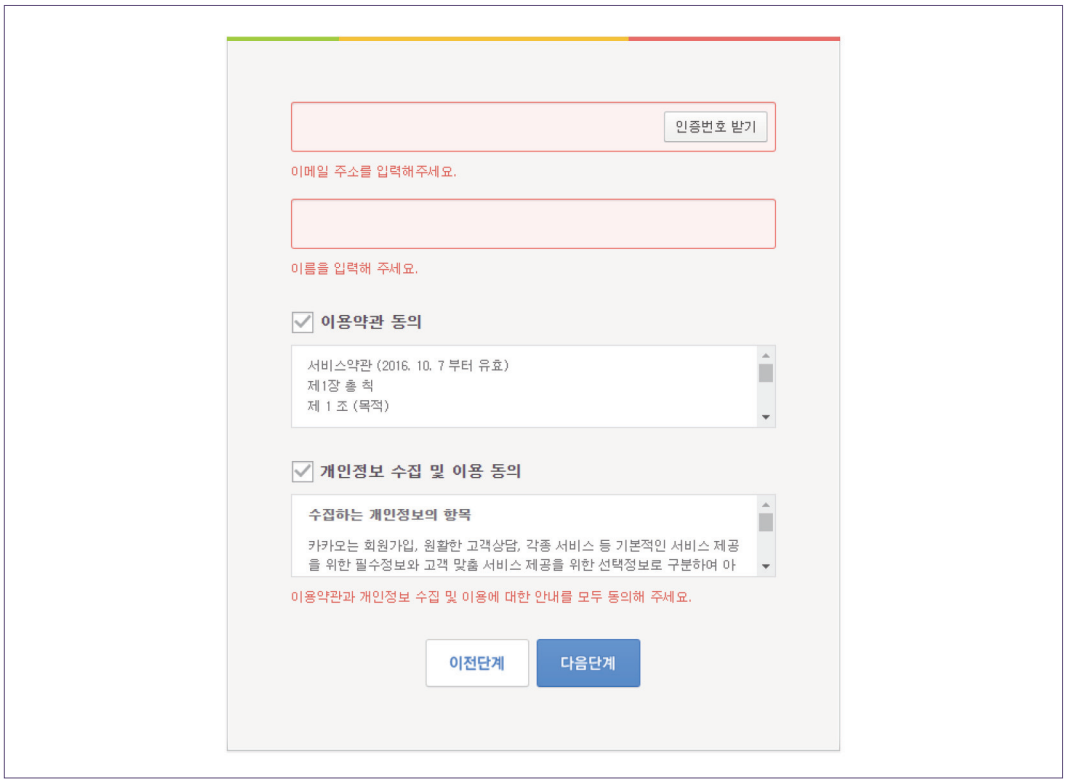


NAVER 스크린리더(NVDA) 음성

[아무것도 입력하지 않고 “가입하기” 버튼 클릭 시]

- 버튼
- 가입하기

에러가 발생한 Daum 회원가입 입력 폼



DAUM 스크린리더(NVDA) 음성

[아무것도 입력하지 않고 “다음단계” 버튼 클릭 시]

- 버튼
- 한글
- 편집창 자동완성

위의 페이지를 실제로 스크린리더로 읽어본 결과 에러가 발생되었을 때 아무 것도 들리지 않는다.

NAVER와 같은 경우는 에러가 발생해도 포커스가 버튼에 유지되고 있고 어떠한 방법으로도 에러가 발생했다는 것을 알려주지 않는다. 에러가 발생한 것인지 버튼이 정상적으로 동작하는지 바로 알기 힘들다.

DAUM과 같은 경우는 에러가 발생한 영역으로 포커스가 이동되지만 에러가 발생했다는 것을 안내하지 않고 포커스만 이동되어 스크린리더 사용자에게 혼란을 가중시킨다.

• WAI-ARIA를 사용해야 하는 이유

비 장애인들이 쉽게 에러 발생을 인지할 수 있는 것처럼 WAI-ARIA 속성을 잘 사용하면 스크린리더 사용자들도 비 장애인처럼 쉽게 폼 서식을 사용할 수 있다. 에러가 발생했을 때 "입력 오류"를 들을 수 있게 하고, 어떤 에러인지 에러메시지를 스크린리더가 읽을 수 있도록 제공하면 되는데 아주 간단한 WAI-ARIA 속성 두 가지만 사용하면 가능하다.

에러가 발생한 것을 바로 인식하기 어려운 경우는 aria-live 속성을 사용하면 스크린리더 사용자도 바로 인지할 수 있다.

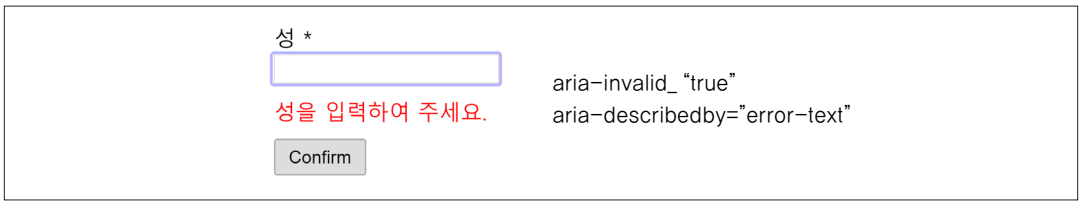
• WAI-ARIA 전체적인 그림과 설명

에러가 발생하기 전에는 기본 마크업으로만 되어 있고, 에러가 발생했을 때는 동적으로 aria-invalid 속성이 true가 되고 aria-describedby 속성이 id와 연결되어 생성된다.

그리고 에러가 발생되지 않으면 aria-invalid 속성이 false가 되거나 삭제되어야 하고 에러메시지와 연결된 aria-describedby 속성도 삭제되어야 한다. 그렇지 않으면 계속 에러가 발생한 것으로 오해할 수 있다.

전체적인 그림을 보면 아래 그림과 같다.

WAI-ARIA를 사용했을 때 폼과 연결된 에러 발생 후 모습



에러가 발생했을 때 에러가 발생한 영역으로 포커스를 이동시키면 아래와 같이 스크린리더가 읽는다.

에러 발생 시 스크린리더(NVDA) 음성

- Confirm 버튼
- 성 * 필수입력항목 편집창 입력 오류 자동완성 성을 입력하여 주세요.

만약, 서식 입력 폼이 여러 개가 같이 존재하고 에러가 여러 폼에서 동시에 발생할 때는 에러 발생한 모든 폼에 aria-invalid 속성과 aria-decribedby 속성이 동적으로 삽입되고, 포커스는 에러가 발생한 첫 번째 폼으로 이동시킨다. 그렇게 하면 스크린리더 가상모드 탐색 시 에러가 발생한 영역을 스크린리더 사용자는 쉽게 알 수 있다.

또한 달력과 같이 특정 폼과 연결할 수 없는 에러가 발생할 때 사용할 수 있는 역할, 속성으로는 가장 많은 스크린리더에서 호환되도록 사용하는 방법으로 role="alert"과 aria-live="assertive"를 사용하는 것이 있다. 페이지 로드 시 이미 해당 속성을 가지고 있는 빈 태그 안에 동적으로 에러 문구 노드를 삽입하면 즉각적으로 스크린리더가 읽는다.

Note.

role="alert"은 동적으로 삽입되는 중요한 정보를 사용자에게 전달할 때 사용하는 역할로 해당 요소 안에 텍스트가 삽입되면, 스크린리더가 읽던 내용을 끊고 삽입된 텍스트를 스크린리더가 즉각적으로 읽게 되므로 주의하여 사용해야 한다.

• WAI-ARIA 속성 정리

에러가 발생했을 때 해당 영역에 aria-invalid 속성을 사용하면 "입력 오류"라고 스크린리더가 읽게 된다. 또한 어떤 에러가 발생했는지 에러메시지를 aria-describedby 속성을 사용하여 연결하고 에러가 발생한 〈input〉 요소로 포커스를 이동시키면 스크린리더가 에러메시지를 읽게 된다.

요소	역할	속성/상태	설명
〈input〉 〈select〉 〈textarea〉		aria-invalid="true/false"	에러가 발생하고 있음을 알림
		aria-describedby="{id}"	에러문구가 갖고 있는 id와 연결하여 어떤 에러가 발생되었는지를 알림
〈div〉 에러 문구	alert	aria-live="assertive"	속성 값은 polite, assertive를 사용할 수 있으며 assertive를 많이 사용 (호환)

• 소스 코드 정리

다음은 〈label〉 요소와 〈input〉 요소로 제작된 일반적인 마크업이다. 에러가 발생하면 삽입될 수 있도록 에러 메시지가 삽입될 빈 태그도 함께 미리 제작한다.

```
<!-- 폼과 연결된 에러 -->
<div>
    <label for="fname">성 <span aria-hidden="true">*</span>
    <span class="offscreen">필수입력항목</span></label>
    <input id="fname" type="text">
    <div class="error" id="error-text"></div>
</div>
<button class="btn">Confirm</button>
<!-- 폼과 연결되지 않는 에러 -->
<div class="error" id="error-text" role="alert" aria-live="assertive"></div>
<button class="btn">Confirm</button>
```

〈input〉 요소에 아무것도 입력하지 않거나 숫자를 입력할 때 에러가 발생한다고 가정하고, 에러를 발생시킨다.

에러가 발생하면 동적으로 에러 메시지가 삽입되고, 에러가 발생한 〈input〉 요소에 aria-invalid 속성을 정의하고, 폼이 에러 메시지의 id 값과 연결되도록 aria-describedby 속성을 삽입한다.

그리고, 에러가 발생한 〈input〉 요소로 포커스를 이동시키면 스크린리더가 에러가 발생했다는 것을 읽어준다.

```
<!-- 폼과 연결된 에러 -->
<div>
    <label for="fname">성 <span aria-hidden="true">*</span>
    <span class="offscreen">필수입력항목</span></label>
    <input type="text" id="fname" aria-invalid="true"
           aria-describedby="error-text">
    <div class="error" id="error-text">성을 입력하여 주세요.</div>
</div>
<button class="btn">Confirm</button>
<!-- 폼과 연결되지 않는 에러 -->
<div class="error" id="error-text" role="alert" aria-live="assertive">
    <span>성 입력 오류</span>
</div>
<button class="btn">Confirm</button>
```

폼과 연결된 에러는 위의 예제처럼 〈input type="text"〉 요소 외에도 라디오버튼, 체크박스, 셀렉트박스 등에도 적용할 수 있고, 폼과 연결되지 않는 에러에는 Live Region 속성으로 모두 응용이 가능하다.

jQuery를 활용하여 폼과 연결된 에러 발생 시 aria-invalid 속성과 aria-describedby 속성이 동적으로 삽입되도록 아래와 같이 자바스크립트를 제작한다.

```
$(document).ready(function() {
    $(".btn").click(function() {
        testInput();
        $('#fname').on('keydown', function() {
            testInput();
        });
    });
    function testInput() {
        var val = $('#fname').val();
        if (val === '') {
            // 아무것도 입력하지 않을 때 에러 발생
            $(".error").html("성을 입력하여 주세요.");
            $("#fname").attr({
                // aria-invalid 속성과 true 값을 삽입
                'aria-invalid': true,
                // 에러 문구와 폼을 aria-describedby속성과 id로 연결
                'aria-describedby': "error-text"
            });
            // 포커스는 입력상자로 이동
            $("#fname").focus();
        } else {
            if ((/[a-zA-Z]+$/).test(val)) {
            // 에러가 발생하지 않으면 aria-invalid와 aria-describedby 속성 삭제
                $("#fname").removeAttr('aria-invalid');
                $("#fname").removeAttr('aria-describedby');
                $(".error").html('');
            } else {
                // 영문 이 외를 입력할 때 에러 발생시킴
                $(".error").html("영문으로 입력하여 주세요.");
                // aria-invalid 속성과 true 값을 삽입
                // 에러 문구와 폼을 aria-describedby속성과 id로 연결
                $("#fname").attr({
                    'aria-invalid': true,
                    'aria-describedby': "error-text"
                });
                $("#fname").focus();
            }
        }
    }
});
```

9. 레이어 팝업(Layer Popup)

• 기존 코드의 문제점들

웹 페이지내에서 시각적으로 팝업 창과 같은 효과를 내도록 같은 페이지 내에서 기존 콘텐츠를 가리고 그 위의 새로운 층에 팝업 창 형태의 콘텐츠 영역을 보이도록 한 경우, 이를 일반적으로 레이어 팝업(Layer Popup)이라고 부른다.

또한 레이어 팝업은 링크나 버튼요소에 클릭 또는 키보드 이벤트가 발생했을 때 현재 페이지에서 다른 페이지로 이동하지 않고 현재 페이지에서 사용자가 정의한 다이얼로그이다. 예를 들어 로그인 페이지, 상세 설명 등에서 사용된다.

레이어 팝업이 사용된 페이지에서 레이어 팝업을 호출한 이후 포커스를 레이어 팝업으로 지정하고, 레이어 팝업이 닫힌 후에는 다시 레이어 팝업을 호출하기 전 포커스로 돌려보내는 작업이 필요하다.

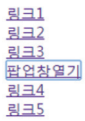
이때, 레이어 팝업에서의 탭키는 순환적으로 링크를 이동해야 하고, Esc키를 사용할 경우는 레이어 팝업을 종료하고 레이어 팝업을 호출하기 전으로 포커스가 이동해야 한다. 이러한 규칙을 갖고 접근성을 고려한 소스를 구현해야 한다.

Note.

레이어 팝업(Layer Popup)을 화면에 띄웠을때 해당 레이어 팝업으로 자바스크립트를 이용하여 초점을 이동할 경우 스크린리더에서는 가상 커서방식이므로 초점 변화의 이동을 인지하지 못하고 이를 읽을 수 없다. 대신 스크린리더의 브라우저모드(윈도우커서)를 사용할 경우 자바스크립트로 강제로 초점을 이동시키면 읽을 수 있다.

레이어 팝업의 키보드 사용순서

❶ 엔터키/스페이스바 사용 시 팝업 생성



❷ 포커스가 팝업으로 이동

❸ 탭키 사용 시 순차적으로 이동

❹ 닫기 버튼 / Esc키 사용 시 팝업 호출 전으로 포커스 이동

또한 콘텐츠의 선형화를 고려한다면 레이어 팝업(Layer Popup) 코드는 팝업창열기 링크 다음에 위치하는 것이 좋다.

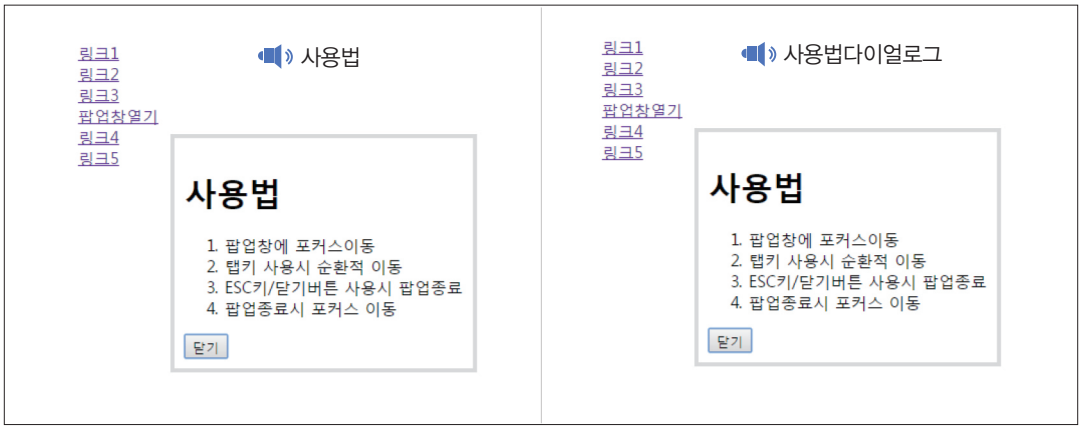
```
<div>
    <a href="#" id="layer_open">팝업창열기</a>
</div>
<div class="layer_area" id="layer">
    <h1 id="dialogTitle" tabindex="0">사용법</h1>
    <ol>
        <li>팝업창에 포커스이동</li>
        <li>탭키 사용시 순환적 이동</li>
        <li>ESC키/닫기버튼 사용시 팝업종료</li>
        <li>팝업종료시 포커스 이동</li>
    </ol>
    <div>
        <input type="button" value="닫기" id="layer_close">
    </div>
</div>
```

• WAI-ARIA를 사용해야 하는 이유

레이어 팝업(Layer Popup)은 접근성 측면에서 보면 키보드 사용자에게 별 다른 문제는 없다. 하지만 스크린리더를 이용하는 사용자에게는 레이어 팝업은 〈div〉 요소의 내용을 읽어주기 때문에 팝업이 발생했는지 정확히 전달하기 어렵다.

이때 WAI-ARIA를 사용하여 레이어 팝업임을 인지할 수 있게 하는 것이 가능하다.

레이어 팝업 음성출력 결과

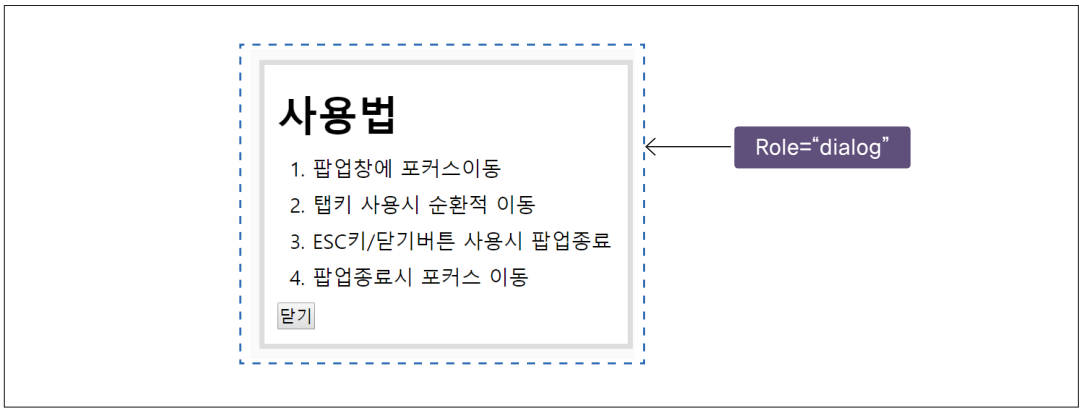


위의 그림 왼쪽은 WAI-ARIA를 적용하지 않은 경우이고 오른쪽은 WAI-ARIA를 적용한 경우이다. 이때 왼쪽의 경우에는 〈h1〉 요소에 있는 내용을 음성 출력하기 때문에 팝업인지 정확히 알 수 없지만 오른쪽 경우는 〈h1〉 요소에 있는 내용과 역할(Role)에 부여된 dialog로 인해 다이얼로그라는 음성을 추가적으로 출력해 주어 레이어 팝업임을 알 수 있다.

• WAI-ARIA 전체적인 그림과 설명

우선 전체적인 WAI-ARIA의 역할(Role) 구조를 살펴보자.

레이어 팝업(Layer Popup)에 대한 WAI-ARIA 구조



WAI-ARIA의 역할(Role), 속성(Property), 상태(State)를 이용하여 스크린리더 사용자로 하여금 해당 UI를 인식할 수 있도록 의미를 부여해보자.

〈div〉 요소에 다이얼로그(dialog) 역할을 role 속성을 사용하여 추가하고, 레이어 팝업을 볼 수 없게 aria-hidden 속성의 값을 true로 설정한다.

또한 레이어 팝업의 제목을 지정할 수 있도록 aria-labelledby="{id값}"을 지정한다. 이때 {id 값}은 〈h1〉 요소의 id="{id 값}"과 일치시키면 스크린리더에서는 〈h1〉 요소의 제목 내용인 "레이어 팝업"을 음성으로 출력 한다.

• WAI-ARIA 속성 정리

각 요소에 적용된 역할(Role)과 속성(Property), 상태(State)를 정리해보면 다음과 같다.

요소	역할	속성/상태	설명
〈div〉	alertdialog		다이얼로그 역할을 정의
		aria-labelledby	레이어 팝업에 대한 제목을 기술해 둔 요소와 연결
		aria-hidden	aria-hidden="true" 해당 요소를 보조기기에서 들리지 않게 함. aria-hidden="false" 해당 요소를 보조기기에서 들리게 함.

• 소스 코드 정리

기존의 레이어팝업 소스를 WAI-ARIA를 이용하여 개선한 코드는 아래와 같다.

```
<!DOCTYPE html>
<html lang="ko">
    <head>
        <meta charset="utf-8">
        <title>Layer Popup</title>
        <link id="size-stylesheet" rel="stylesheet" type="text/css" />
        <script src="http:// code.jquery.com/jquery-1.10.1.min.js"></script>
        <script src="http:// code.jquery.com/jquery-migrate-1.2.1.min.js">
        </script>
        <script type="text/javascript">
            $(document).ready(function() {
                $('#layer').keyup(function(e){
                    var keyCode = e.keyCode || e.which; // 키보드 코드값
                    if(keyCode == 27){// ESC 키
                        $("#layer").attr("aria-hidden","true");
                        $("#layer_open").attr("tabindex","0");
                        $("#layer_open").focus() ;
                    }
                });
                $('#layer_open').keyup(function(e){
                    var keyCode = e.keyCode || e.which;
                    if(keyCode == 13 || keyCode ==32) {// 엔터키또는 스페이바키
                        $("#layer").attr("aria-hidden","false");
                        $("#layer_open").attr("tabindex","-1");
                        $("#layer h1").focus();
                    }
                });
                $('#layer_close').keyup(function(e){
                    var keyCode = e.keyCode || e.which;
                    if(keyCode == 13) {// 엔터키
                        $("#layer").attr("aria-hidden","true");
                        $("#layer_open").attr("tabindex","0");
                        $("#layer_open").focus() ;
                        // e.preventDefault();
                    }
                });
```

```
            $('#layer_close').keydown(function(e){
                var keyCode = e.keyCode || e.which;
                if (e.shiftKey  && keyCode == 9 ) { // shift+tab 키
                    $(this).prev().focus();// 이전 링크로 커서이동
                }else if(keyCode == 9){// 탭키
                    e.preventDefault();// 탭키의 기본기능 삭제
                    $('#layer h1').focus();// 첫번째 링크로 이동
                }
            });
            $('#layer h1').keydown(function(e){
                var keyCode = e.keyCode || e.which;
                if (keyCode == 9 && e.shiftKey) {// shift+tab 키
                    e.preventDefault();
                    $("#layer").attr("aria-hidden","true");
                    $("#layer_open").attr("tabindex","0");
                    $("#layer_open").focus() ;
                }
            });
            $('#layer_open').mousedown(function(e){
                $('#layer').show();
            });
            $('#layer_close').mousedown(function(e){
                $('#layer').hide();
            });
        });
    </script>
    <style>
        .layer_area {
            position : absolute;
            left : 100px;
            top :100px;
            background : #fff;
            padding : 20px;
            border : 4px solid #ddd;
        }
        div[aria-hidden='true'] {
            display: none;
        }
    </style>
</head>
<body>
```

```
        <div><a href="#">링크1</a></div>
        <div><a href="#">링크2</a></div>
        <div><a href="#">링크3</a></div>
        <div>
            <a href="#" id="layer_open">팝업창열기</a>
        </div>
        <div class="layer_area" id="layer" aria-hidden="true" role="dialog"
             aria-labelledby="dialogTitle">
             <h1 id="dialogTitle" tabindex="0">레이업 팝업</h1>
             <ol>
                 <li>팝업창에 포커스이동</li>
                 <li>탭키 사용시 순환적 이동</li>
                 <li>ESC키/닫기버튼 사용시 팝업종료</li>
                 <li>팝업종료시 포커스 이동</li>
             </ol>
             <div>
                 <input type="button" value="닫기" id="layer_close">
             </div>
        </div>
        <div><a href="#">링크4</a></div>
        <div><a href="#">링크5</a></div>
    </body>
</html>
```

• 키보드 인터랙션

키보드	인터랙션
탭키	다음 링크로 이동. 마지막 링크에서 탭키 사용 시 처음 링크로 이동
Esc키	레이어 팝업(Layer Popup)을 숨기고 레이어 팝업(Layer Popup)을 호출하기 전 포커스로 이동
Shift+탭키	이전 링크로 이동. 첫번째 링크에서 Shift + 탭키 사용 시 레이어 팝업(Layer Popup)을 숨기고 레이어 팝업(Layer Popup)을 호출하기 전 포커스로 이동

10. 툴팁(Tooltip) UI

• 기존 코드의 문제점들

툴팁(Tooltip)은 텍스트, 이미지 위에 마우스 커서가 올라갈 때 말 풍선처럼 보충 설명을 표시하는 UI 컴포넌트로 공간이 협소하여 미처 표시하지 못한 내용을 마우스 인터랙션에 따라 화면에 표시할 수 있어 유용하게 사용된다.

section.cafe.naver.com 구독게시판 새 글소식「툴팁」

section.blog.naver.com 핫 토픽 목록「툴팁」

툴팁을 구현하는 방법은 다양한데 가장 간단한 방법은 HTML 요소에 title 속성을 추가하고 표시할 텍스트 내용을 값으로 제공하면 된다. 하지만 이와 같은 방법은 간단한 툴팁을 제공할 뿐, 위의 예에서 살펴본 것과 같은 복잡한 구조와 유려한 디자인을 제공할 수는 없다.

CSS를 활용하면 위의 예와 같은 툴팁 제작이 가능해진다. CSS의 가상(유사) 클래스인 :hover를 활용하면 마우스 커서가 올라간 상태 처리를 이용하여 보이지 않던 툴팁을 화면에 표시해줄 수 있다. 이때 마우스를 사용하지 못하는 상황의 사용자를 고려해 키보드를 통해 툴팁에 접근할 수 있도록 :hover와 함께 가상 클래스 :focus도 사용해주어야 한다.

다음은 툴팁 제작을 위한 HTML과 CSS 코드이다.

```
<a href="..." target="_blank" class="tooltip-btn">
    <img src="..." width="129" height="26" alt="구독게시판 새 글소식">
</a>
<div class="tooltip-content">
    <p>즐겨찾는 게시판의 새소식을<br>네이버me에서 모아보세요!</p>
</div>
```

```
/* 초기 로딩 시, 툴팁 내용은 화면에서 감춰짐 */
.tooltip-content {  display: none; }
/* 버튼에 마우스가 올라가거나, 키보드 포커싱 상태가 되면 화면에 보여짐 */
.tooltip-btn:hover + .tooltip-content,
.tooltip-btn:focus + .tooltip-content {
    display: block;
}
```

더 나아가 자바스크립트를 사용하면 기본적인 마우스 이벤트뿐만 아니라, 마우스 커서를 이동할 때마다 툴팁에 다양한 디자인을 가미할 수 있다. 하지만 이와 같은 시각적 효과를 필요로 하는 디자인 컨셉이 아니라면, CSS 만으로도 툴팁 구현은 충분하고 손쉽게 구현할 수 있다.

문제는 앞서 기술한 방법으로 툴팁을 구현해 키보드 사용자를 위한 접근성은 갖췄다 하더라도, 스크린 리더 사용자의 경우 해당 요소에 포커스가 되었을 때 툴팁 내용이 있다고 안내받거나 툴팁 내용을 읽어주지 않아 정보를 제공받지 못하는 문제가 있다.

링크 요소에 포커스된 상태의 「툴팁」

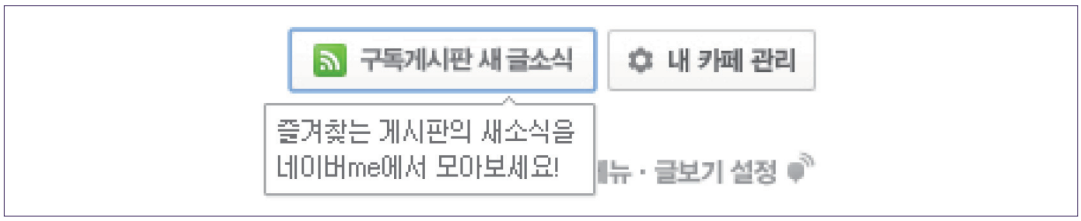
스크린리더 음성(NVDA)

팬택 스카이 아임백(IM-100) 공개! 선택 받을 수 있을까? 링크

• WAI-ARIA를 사용해야 하는 이유

앞서 논의된 문제를 해결하기 위한 손쉬운 방법을 WAI-ARIA는 제공하고 있다. 마우스 커서가 올라가거나, 키보드 포커싱 상태가 되는 요소에 툴팁에 해당하는 요소의 설명임을 명시해주고, 툴팁에 해당하는 요소는 툴팁 역할을 부여해주면 툴팁 내용을 읽어준다.

구독게시판 새 글소식 링크 요소에 포커스된 상태의 「툴팁」



스크린리더 음성(NVDA)

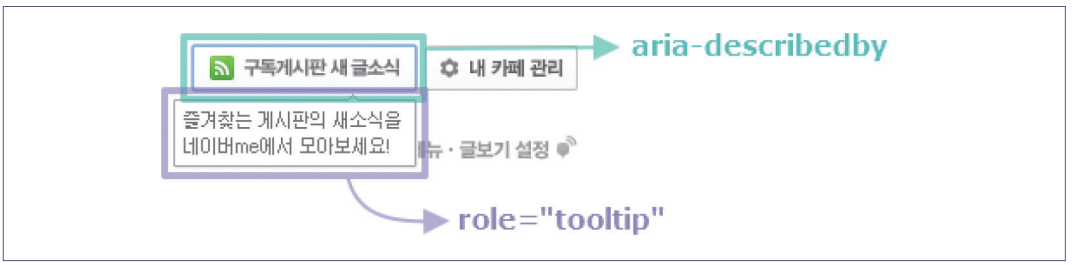
구독게시판 새 글소식 그래픽 링크

즐겨찾는 게시판의 새소식을 네이버me에서 모아보세요!

• WAI-ARIA 전체적인 그림과 설명

툴팁 UI 컴포넌트는 크게 툴팁을 띄우는 요소와 툴팁 요소로 나뉜다. 툴팁을 띄우는 요소에 연결된 툴팁이 어떤 요소인지를 명시해주고, 툴팁 요소에는 툴팁 역할과 툴팁으로서 식별 가능한 id 속성을 설정해주면 툴팁 UI 컴포넌트의 접근성을 향상시킬 수 있다.

툴팁 UI 컴포넌트에 설정된 WAI-ARIA 구조



• WAI-ARIA 속성 정리

앞서 살펴본 툴팁 UI 컴포넌트에 적용한 WAI-ARIA 속성을 정리해보자.

요소	역할	속성/상태	설명
〈a〉 〈button〉 〈input〉		aria-describedby=""	툴팁 요소 식별자(id) 설정
〈div〉 〈span〉	role="tooltip"		툴팁 역할을 설정
		id=""	툴팁 식별자(id) 설정

• 소스 코드 정리

툴팁 UI 컴포넌트에 접근성 향상을 위한 WAI-ARIA 속성을 추가해보자. 앞서도 이야기 했지만, 적용 방법은 매우 쉽고 간단하다. 다음과 같이 각 요소에 WAI-ARIA 속성을 추가해주면 된다.

```
<a href="..."  target="_blank" class="tooltip-btn"
    aria-describedby="notice-tip" >
     <img src="..." width="129" height="26" alt="구독게시판 새 글소식">
</a>
<div id="notice-tip" class="tooltip-content" role="tooltip">
     <p>즐겨찾는 게시판의 새소식을<br>네이버me에서 모아보세요!</p>
</div>
```

11. 필수 입력 항목(Required)

• 기존 코드의 문제점들

웹 사이트에서 사용자로부터 데이터를 전송 받기 위해 온라인 서식을 사용하게 되는데, 사용자로부터 입력을 받기 위한 폼 관련 요소(form-associated element) 중 일부는 필수로 입력을 요하는 경우가 종종 있다.

온라인 서식 예

위와 같은 예에서 각 폼 요소는 모두 필수로 입력이 되어야 하는 항목들이지만, 해당 항목들이 필수 항목임을 인지할 수 있는 장치가 없다면 사용자는 폼 검증(form validation)이 이루어지는 시점에 도달하기 이전까지는 현재 입력 중인 항목이 필수 입력 항목인지의 여부를 알기 어렵다.

때문에 이를 해소하기 위한 방법으로 대부분 〈label〉 요소 혹은 별도의 영역에 필수 항목임을 인지 할 수 있는 표기를 제공하거나, HTML5 문서의 경우에서는 required 속성을 이용하고 있다.

〈label〉 요소에 특수문자(*)로 필수 항목을 표한한 마크업

```
<span>
    <strong>*</strong>표시 항목은 필수 입력 항목입니다
</span>
...
<li>
    <label class="hidden-accessible" for="userId">* 아이디</label>
    <input type="text" id="userId" name="userId" maxlength="12"
           placeholder="아이디 (ID) *">
    <span id="form-guide">6~12자 영문, 숫자로 입력 해주세요.</span>
</li>
<li>
   <label class="hidden-accessible" for="userName">이름</label>
   <input type="text" id="userName" name="userName" maxlength="20"
          placeholder="이름">
</li>
...
<li>
    <label class="hidden-accessible" for="phoneNumber">휴대폰 번호</label>
    <input type="text" id="phoneNumber" name="phoneNumber" maxlength="13"
           placeholder="휴대폰 번호">
</li>
<li>
    <label class="hidden-accessible" for="password">* 비밀번호</label>
    <input type="password" id="password" name="password" maxlength="12"
           placeholder="비밀번호 *" aria-describedby="pwdToolTip" >
    <span id="pwdToolTip">
        영문, 숫자를 혼합하여 6~12자로 입력 해주세요.
        (특수문자는 .!@#$% 만 허용됩니다.)
    </span>
</li>
```

〈input〉 요소에 required 속성을 지정한 마크업

```
<span aria-hidden="true">
    <strong>*</strong>표시 항목은 필수 입력 항목입니다
</span>
...
<li>
    <label class="hidden-accessible" for="userId">아이디</label>
    <input type="text" id="userId" name="userId" maxlength="12"
           placeholder="아이디 (ID) *" required >
    <span id="form-guide">6~12자 영문, 숫자로 입력 해주세요.</span>
</li>
<li>
    <label class="hidden-accessible" for="userName">이름</label>
    <input type="text" id="userName" name="userName" maxlength="20"
           placeholder="이름">
</li>
...
<li>
    <label class="hidden-accessible" for="phoneNumber">휴대폰 번호</label>
    <input type="text" id="phoneNumber" name="phoneNumber" maxlength="13"
           placeholder="휴대폰 번호">
</li>
<li>
    <label class="hidden-accessible" for="password">비밀번호</label>
    <input type="password" id="password" name="password" maxlength="12"
           placeholder="비밀번호 *" aria-describedby="pwdToolTip" required >
    <span id="pwdToolTip">
        영문, 숫자를 혼합하여 6~12자로 입력 해주세요.
        (특수문자는 .!@#$% 만 허용됩니다.)
    </span>
</li>
```

많은 웹 사이트들이 필수 항목을 표기하기 위해 전자의 경우와 같이 "필수"를 상징하는 * 특수 문자를 표기하는 것과 같은 방법을 사용하고 있다. 하지만 불행히도, 일부 스크린리더는 기본 설정으로 특수 문자를 읽지 않도록 되어 있고 특수 문자를 읽도록 설정한다 하여도 *를 읽지 못하는 경우가 있다.

때문에 이러한 현상을 피하기 위한 또 다른 대안, 예를 들어 스크린리더 사용자에게만 제공되는 숨김 텍스트 등이 필요하게 되고 이는 제법 번거로운 일이 될 수 있다.

HTML5 문서라면 사정이 조금 다르다. HTML5가 충분히 지원되는 유저 에이전트에서라면 required 속성(Attribute)이 앞의 문제를 좀 더 간소화 시켜 줄 수 있기 때문이다. HTML5에서의 required 속성(Attribute)은 스크린리더에게 "필수"라는 의미를 전달해 주도록 명세에 정의가 되어 있다. 따라서 required 속성(Attribute)의 사용은 스크린리더 사용자에게 "필수"의 의미를 전달하는 음성 출력을 내 줄 것이라 기대할 수 있다.

하지만, 여전히 문제가 존재하게 되는데, HTML5의 required 속성을 충분히 지원하지 못하는 유저 에이전트(예를 들어, Internet Explorer 9 등)가 있다는 점이고, 모든 웹 페이지들이 HTML5로 작성된다는 보장이 없다라는 것이며, 유저 에이전트가 자체적으로 required된 항목들에 대해 폼 검증을 일으켜 노출되는 말풍선 등을 디자인 측면의 이슈로 사용하기를 꺼리기 때문에 잘 활용하지 않는 경우가 많다라는 점이다.

• WAI-ARIA를 사용해야 하는 이유

WAI-ARIA의 aria-required 속성(Property)을 사용한다면 HTML5의 required 속성(Attribute) 적용의 제약사항이나 스크린리더 사용자를 위한 숨김 텍스트를 제공해야 하는 번거로움을 다소 해소할 수 있다.

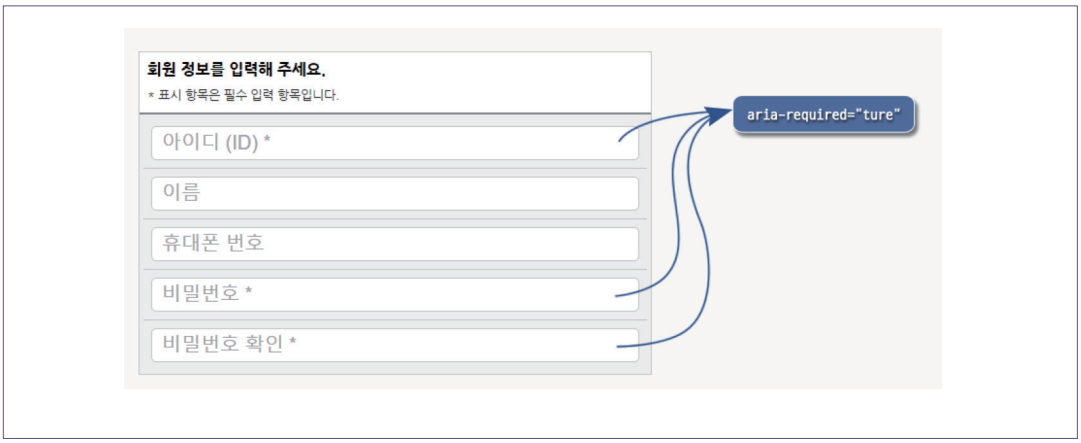
aria-required 속성(Property)은 HTML5의 required 속성(Attribute)의 의미에 대응되며, aria-required 속성의 값을 true로 설정하게 되면 스크린리더는 이 속성(Property)이 true로 설정된 요소(Element)에 대해 필수 입력 항목으로 인식하게 된다.

즉, 문서 타입이 HTML5가 아닌 HTML4.01 혹은 XHTML1.0이라 하더라도, 혹은 유저 에이전트가 required 속성(Attribute)을 지원하지 않는다 하더라도, 단순히 aria-required 속성(Property)을 설정함으로 스크린리더로 하여금 해당 요소를 필수 항목 혹은 비 필수 항목으로 인식 할 수 있도록 제공이 가능하다.

• WAI-ARIA 전체적인 그림과 설명

적용 될 WAI-ARIA 의 구조를 그림으로 살펴보자. 예제에서는 WAI-ARIA 적용의 차이를 명확히 하기 위해 required 속성(Attribute)도 적용되지 않은 경우와 여기에 WAI-ARIA를 적용한 경우를 비교해 두었다.

WAI-ARIA 상태(State) 구조



단지 필수 항목으로 설정할 요소(Element)에 aria-required="true"를 설정하기만 하면 되고, 이렇게 aria-required 속성이 적용 되었을 때 스크린리더는 다음과 같은 변화를 보여준다.

WAI-ARIA 적용에 따른 스크린리더 음성 출력 차이

WAI-ARIA 적용 이전	WAI-ARIA 적용 이전
NVDA 음성 출력 뷰어 회원 정보를 입력해 주세요. * 표시 항목은 필수 입력 항목입니다. 그루핑 목록 아이디 * 편집창 빈줄 이름 편집창 빈줄 휴대폰 번호 편집창 빈줄 편집창 보호됨 빈줄 비밀번호 확인 * 편집창 보호됨 빈줄	NVDA 음성 출력 뷰어 회원 정보를 입력해 주세요. * 표시 항목은 필수 입력 항목입니다. 그루핑 입력 오류 목록 아이디 * 편집창 입력 오류 필수입력사항 빈줄 이름 편집창 빈줄 휴대폰 번호 편집창 빈줄 편집창 보호됨 입력 오류 필수입력사항 빈줄 비밀번호 확인 * 편집창 보호됨 입력 오류 필수입력사항 빈줄

각 요소에 적용되는 WAI-ARIA 속성을 정리해보면 다음과 같다.

요소	역할	속성/상태	설명
〈input〉		aria-required	필수 항목일 경우 true, 필수 항목이 아닐 경우 false

• 소스 코드 정리

이제, 앞서 설명한 대로 aria-required 속성(Property)을 설정하는 방법을 보도록 하자.

앞서 설명한 바와 같이 해당 요소에 단순히 aria-required="true"를 삽입하는 것으로 족하다.

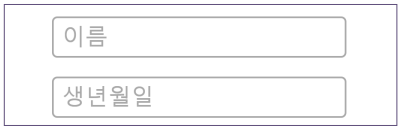
```
<form action="#" name="usrInfo" id="usrInfo" method="post">
    <fieldset>
        <legend>
            회원 정보를 입력해 주세요.
            <span class="require">
                * 표시 항목은 필수 입력 항목입니다.
            </span>
        </legend>
        <ul>
            <li>
                <input type="text" id="userId2" name="userId2"
                       maxlength="12" aria-required="true">
                <label for="userId2">아이디 *</label>
            </li>
            <li>
                <input type="text" id="userName2" name="userName2"
                       maxlength="20" >
                <label for="userName2">이름</label>
            </li>
            <li>
                <input type="text" id="phoneNumber2" name="phoneNumber2"
                       maxlength="13" >
                <label for="phoneNumber2">휴대폰 번호</label>
            </li>
            <li>
                <input type="password" id="userPwd2" name="userPwd2"
                       maxlength="12" aria-required="true">
                <label for="userPwd2">비밀번호 *</label>
            </li>
            <li>
                <input type="password" id="userPwdRe2" name="userPwdRe2"
                       maxlength="12" aria-required="true">
                <label for="userPwdRe2">비밀번호 확인 *</label>
            </li>
        </ul>
    </fieldset>
</form>
```

12. 플레이스 홀더(Placeholder)

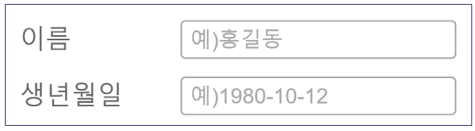
• 기존 코드의 문제점들

플레이스 홀더(Placeholder)란 HTML5에서 적용 가능한 입력 서식(〈input〉, 〈textarea〉 요소 등)에 사용할 수 있는 속성으로 한 단어나 짧은 문구로 이루어진 '힌트'를 나타 낼 때 사용한다. 이때 placeholder 속성을 〈label〉 요소를 대체할 목적으로 사용해서는 안 된다.

placeholder의 잘못된 사용 예시



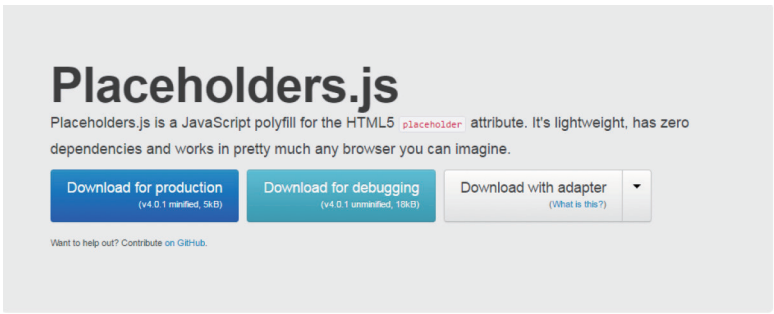
placeholder의 올바른 사용 예시



또한, 크로스 브라우징 이슈로 HTML5를 지원하지 않는 브라우저(IE8 이하 버전 등)에서는 별도의 작업이 필요하다. 이런 작업을 하기 위해서는 모든 브라우저에서 사용할 수 있도록 만든 Placeholder Polyfill을 사용 하거나 직접 개발해야 한다.

아래의 사이트(http://jamesallardice.github.io/Placeholders.js/)에 이동하여 소소를 다운받아 실행해 보면 HTML5를 지원하지 않는 브라우저에서도 플레이스 홀더(Placeholder) 기능을 사용할 수 있다.

Placeholder를 위한 자바스크립트 Polyfill



Note.

플레이스 홀더(Placeholder)는 접근성 관점에서 보면 자체적인 이슈보다 명도대비 관련 이슈가 더 많다. 일반적으로 웹브라우저가 플레이스 홀더의 텍스트를 표시할 때 반투명 색상으로 제공하기 때문에 저시력 사용자의 경우 내용 식별이 어려울 수 있기 때문이다. 이를 개선하기 위해 플레이스 홀더의 텍스트 색상은 배경과 명도대비가 최소 4.5:1을 준수할 수 있도록 하는 것이 필요하다.

• WAI-ARIA를 사용해야 하는 이유

아래의 코드는 value 속성을 사용한 예제 샘플로 스크린리더에서 value 속성에 명시된 내용을 읽어주지 않고 있는 것을 볼 수 있다.

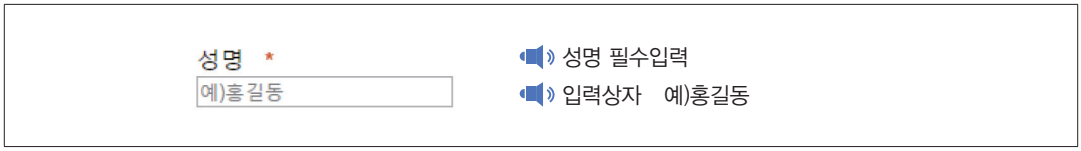
```
<label for="name">성명
    <span class="require-ico"> 필수입력 </span>
</label>
<div>
    <span class="visuallyHidden" id="placeholder-name">예)홍길동</span>
    <input type="text" name="name" id="name" class="inputs" value="예)홍길동" >
</div>
```

value 속성을 사용 시 스크린리더의 음성출력

다음은 WAI-ARIA의 aria-describedby 속성을 사용하여 스크린리더에게 '예) 홍길동' 이라는 문장을 읽을 수 있도록 연결해 보자.

```
<label for="name">성명
    <span class="require-ico"> 필수입력 </span>
</label>
<div>
    <span class="visuallyHidden" id="placeholder-name">예)홍길동</span>
    <input type="text" name="name" id="name" class="inputs"
           aria-describedby="placeholder-name">
</div>
```

aria-describedby 속성을 사용 시 스크린리더의 음성출력



Note.

aria-labelledby 속성을 사용하면 〈label〉 요소에 명시되어 있는 '성명'을 무시하므로 주의해야 한다.

• WAI-ARIA 속성 정리

각 요소에 적용된 역할(Role)과 속성(Property), 상태(State)를 정리해보면 다음과 같다.

요소	역할	속성과 상태	설명
〈input〉		aria-describedby	placeholder의 기능을 구현하기 위한 텍스트를 연결

• 소스 코드 정리

WAI-ARIA를 이용하여 플레이스 홀더(placeholder) 기능을 구현한 소스는 아래와 같다.

```
<!DOCTYPE html>
<html lang="ko">
    <head>
        <meta charset="utf-8">
        <title>PlaceHolder(ARIA)</title>
        <link id="size-stylesheet" rel="stylesheet" type="text/css" />
        <script src="http:// code.jquery.com/jquery-1.10.1.min.js"></script>
        <script src="http:// code.jquery.com/jquery-migrate-1.2.1.min.js">
        </script>
        <script type="text/javascript">
            $(document).ready(function(){
                var value = $("#placeholder-name").text();
                  // <span> 요소에 있는 텍스트
```

```
            $("#name").val(value);
              // <input> 요소의 value값으로 <span>요소의 텍스트를 설정
            $('#name').focusin(function(){// 포커스가 들어왔을 때
                if($( "#name" ).val() == value){
                  // 텍스상자의 내용과 placeholder의 내용이 같으면
                    $("#name").css("color","#000000");
                      // 입력색상으로 변경(placeholder 기능)
                    $('#name').val("");
                      // 기존에 있던 내용을 모두 삭제(placeholder 기능)
                }
            });

            $("#name").focusout(function(){// 포커스가 나갔을 때
               if($( "#name" ).val()==""){// 내용이 비어있다면
                    $("#name").val(value);// "예)홍길동으로 설정
                    $("#name").css("color","#777");
                      // placeholder 색상으로 설정
                    $("#name").css("opacity","1");
               }else{// 내용이 비어 있지 않다면
                    $("#name").css("color","#000000");
                      // 입력상자 색상으로 설정
               }
            });
        });
    </script>
    <style>
        .visuallyHidden {
            height: 0;
            line-height: 0;
            overflow: hidden;
            position: absolute;
            text-indent: -9999px;
            width: 0;
        }
        .require-ico {
            background: url('./ico-required.png') no-repeat 100% 50%;
            width: 15px;
            height: 15px;
            display: inline-block;
```

```
                overflow: hidden;
                line-height: 200rem;
            }
            .inputs {
                color: #777;
                opacity: 1;
            }
        </style>
    </head>
    <body>
        <label for="name">성명
            <span class="require-ico"> 필수입력 </span>
        </label>
        <div>
            <span class="visuallyHidden"
                  id="placeholder-name" >예)홍길동</span>
            <input type="text" name="name" id="name" class="inputs"
                   aria-describedby="placeholder-name" >
        </div>
    </body>
</html>
```

위에서 제공된 소스코드를 기준으로 스크린리더에서는 다음과 같이 읽게된다.

스크린리더(NVDA) 음성

- 성명 필수입력
- 필수입력
- 편집창 예)홍길동

13. 타임 세션(Time Limit)

• 기존 코드의 문제점들

인터넷뱅킹을 사용해 본 이들이라면 다음과 같은 레이어 팝업을 접해 본 경험이 한 번 정도는 있을 것이다.

제한 시간 내에 반응해야 하는 콘텐츠

비단 인터넷뱅킹뿐 아니라, OTP(One Time Passowrd) 코드 입력 등, 이러한 시간 제한이 존재하는 콘텐츠의 경우에는 사용자가 이 제한에 대한 내용을 쉽게 인지하고 제한 시간을 연장하거나 이를 회피할 수 있는 수단이 제공되어야 한다.

다른 여러 가지 형태의 콘텐츠들이 존재하지만, 여기서는 그림과 같이 유효 시간 내에 사용자가 상호작용을 해야 하는 레이어 팝업 형태의 콘텐츠에 대해서 보도록 하자.

위 사이트의 경우, 해당 레이어 팝업이 화면에 뜰 때 다음과 같은 코드를 마크업의 최 상단에 두고 강제 포커스를 줌으로 사용자가 이 팝업의 내용을 먼저 확인하도록 제공했다.

```
<div class="pop-content" id="popContent">
    <a class="pop-focus-anchor" href="#none">
        내부 팝업이 제공되었습니다. 계속해서 내용을 확인해 주세요.
    </a>
    <div class="pop-inner" id="auto_logout_msg">
        <div class="ly-body">
            <div class="ui-pop-outer">
            </div>
            <div class="auto-logout-tw" id="login-area-wrap">
                <div class="logout-info-tw">
                    <dl>
                        <dt class="title">
                            <img alt="자동로그아웃안내" src="...">
                        </dt>
                        <dd class="twenty">
                            자동 로그아웃 남은시간: <em id="cntMsg">20</em>초
                        </dd>
                        <dd class="content">
                            고객님의 안전한 금융거래를 위해 로그인 후<br>
                            약 <span id="totalTime">10</span>분 동안 서비스
                            이용이 없어<br> 로그아웃 됩니다.
                        </dd>
                        <dd class="content02">
                            로그인 시간을 연장하시겠습니까?
                        </dd>
                        <dd>
                            <span>
                               <a href="#"
                                  onclick="__clickYes(); return false;">
                                  연장하기
                               </a>
                            </span>
                            <span>
                               <a href="#"
                                  onclick="__clickNo(); return false;">
                                  로그아웃
                               </a>
                            </span>
                        </dd>
                    </dl>
                </div>
            </div>
        </div>
    </div>
</div>
```

어느 정도 접근성을 제공한 듯 보이지만, 여기에는 몇 가지 부족한 부분이 발생된다.

첫째는, 강제로 포커스가 되었다 하더라도 "내부 팝업이 제공되었습니다. 계속해서 내용을 확인해 주세요." 까지 만을 한 번에 읽어주기 때문에, 처음부터 이것이 제한 된 시간 내에 무언가 반응을 해야만 하는 것과 관련된 팝업인지 인식하기 어려운 문제가 있다.

둘째로는, 가상커서로 문서를 읽어 나가거나 탭키를 이용한 포커스 이동 시 해당 레이어 팝업을 벗어나는 경우가 발생된다라는 점이다.

이와 같은 경우에는 비 스크린리더 사용자가 화면에서 이용하는 것과 같이, 스크린리더 사용자가 레이어 팝업을 벗어나 다른 콘텐츠를 읽는 행동이 사전에 방지되어야 한다.

• WAI-ARIA를 사용해야 하는 이유

물론, 앞서 마크업 되어 있는 방법과 같이 해당 레이어에 강제로 포커스를 주어 어쨌든 이 팝업 내용을 읽어보라고 안내해 주는 것은 그렇지 않은 것 보다는 충분히 접근성을 보장하고 있다.

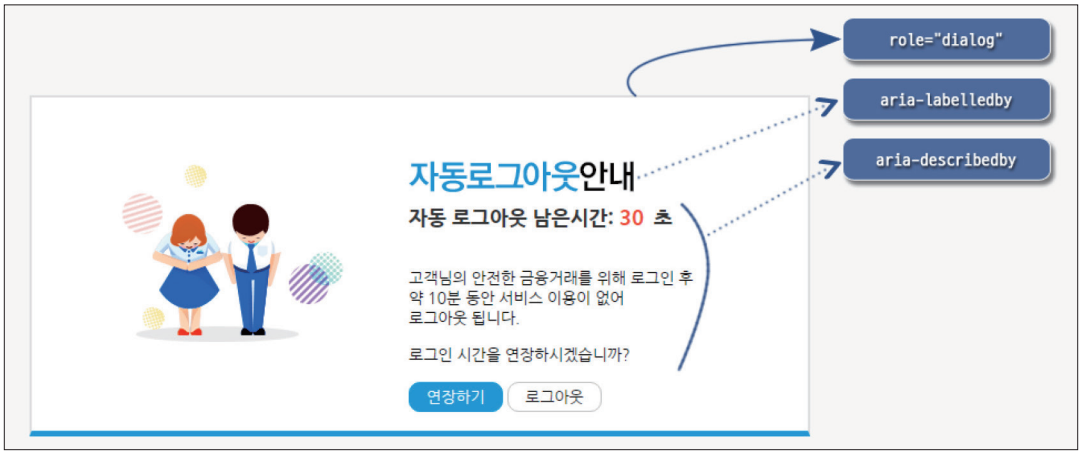
하지만, WAI-ARIA의 aria-labelledby와 aria-describedby 속성(Property)을 사용하게 되면 해당 요소에 접근 시 각 속성에 연결된 텍스트들을 읽을 때, 이 영역이 어떤 제목의 어떤 내용인지를 쉽게 인지할 수 있도록 제공이 가능하다. 이로 인해 좀 더 좋은 접근성을 보장할 수 있게 된다.

또한, dialog 역할(Role)을 제공해 줌으로 현재 읽히는 영역이 대화상자임을 인식하게 할 수 있고, 스크린리더의 가상커서는 이 대화상자를 빠져나가지 않게 되어 해당 레이어 팝업을 벗어나 다른 콘텐츠를 읽는 것을 방지해 줄 수 있게 된다. (단, 포커스 이동은 빠져 나갈 수 있기 때문에 포커스 이동에 대한 별도의 처리는 필요하다.)

• WAI-ARIA 전체적인 그림과 설명

전체적인 WAI-ARIA의 역할(Role) , 속성(Property) 구조를 그림으로 살펴보자.

WAI-ARIA 역할(Role)과 속성(Property) 구조



먼저, 레이어 팝업의 가장 바깥쪽 요소에 dialog 역할(Role)을 부여한다. 여기에 레이어 팝업의 제목이 되는 텍스트를 담는 요소의 id 값을 aria-labelledby 속성(Property) 값으로 연결하여 제목 관계를 설정해 주고, 내용에 해당하는 요소의 id 값을 aria-describedby 속성(Property) 값으로 연결하여 이 요소에 포커싱 될 때 제목과 내용이 한 번에 읽어질 수 있도록 제공한다.

이렇게 WAI-ARIA가 적용 되었을 때, 해당 레이어 팝업에 접근 되는 순간에 대해 스크린리더는 다음과 같은 변화를 보여준다.

WAI-ARIA 적용에 따른 스크린리더 음성 출력 차이

WAI-ARIA 적용 이전	WAI-ARIA 적용 이전
NVDA 음성 출력 뷰어 내부 팝업이 제공되었습니다. 계속해서 내용을 확인해 주세요. 링크 그래픽 자동 로그아웃 안내 자동 로그아웃 남은시간: 52초 고객님의 안전한 금융거래를 위해 로그인 후 약 10분 동안 서비스 이용이 없어 로그아웃 됩니다. 로그인 시간을 연장하시겠습니까? 버튼 연장하기 버튼 로그아웃	NVDA 음성 출력 뷰어 자동 로그아웃 안내 대화상자 자동 로그아웃 남은시간: 60초 고객님의 안전한 금융거래를 위해 로그인 후 약 10분 동안 서비스 이용이 없어 로그아웃 됩니다. 로그인 시간을 연장하시겠습니까? 연장하기 버튼

하이라이트 된 부분이 레이어 팝업이 노출 되었을 때 스크린리더가 한 번에 읽어주는 부분이다.

WAI-ARIA 적용 이전에는 내용을 정확히 알기 위해 사용자의 탐색이 필요한 반면, WAI-ARIA가 적용된 이후에는 사용자의 탐색 행위 없이 시간을 다투는 중요한 정보가 한 번에 사용자에게 전달 되는 것을 확인 할 수 있다.

• WAI-ARIA 속성 정리

각 요소에 적용되는 역할(Role)과 속성(Property)을 정리해보면 다음과 같다

요소	역할	속성과 상태	설명
div	dialog		응용프로그램 대화상자 또는 창을 표기
		aria-labelledby	대화상자의 레이블을 설정
		aria-describedby	대화상자에 대한 설명 설정

• 소스 코드 정리

이제, 앞서 설명한 대로 WAI-ARIA를 적용 시켜 보도록 하자.

참고로, 예제에서는 쉬운 설명을 위해 레이어 팝업이 이미 문서 내에 마크업 되어 있고, CSS로 숨겨져 있다는 가정으로 진행한다.

우선 레이어에 해당하는 요소에 role="dialog"를 설정한다.

```
<div class="layerpop" id="popContent" role="dialog">
    <div class="pop-inner">
       <img src="./images/txt_auto_logout_info.gif"
            alt="자동 로그아웃 안내" >
        <div>
            <strong>자동 로그아웃 남은시간:
                <em id="cntMsg">60</em>초
            </strong>
            <p>
                고객님의 안전한 금융거래를 위해 로그인 후 약 10분 동안 서비스 이용이 없어
                로그아웃 됩니다.
            </p>
            <p>
                로그인 시간을 연장하시겠습니까?
            </p>
        </div>
        <button class="extend" type="button">연장하기</button>
        <button type="button">로그아웃</button>
    </div>
</div>
```

레이어 팝업의 제목과 내용을 관계 짓기 위해 각각에 대응되는 요소에 id 속성(Attribute)을 부여하고, 이 id 값을 aria-labelledby, aria-describedby 속성(Property)의 값으로 설정한다.

```
<div class="layerpop" id="popContent" role="dialog"
     aria-labelledby="pop-title" aria-describedby="pop-content">
    <div class="pop-inner">
       <img src="./images/txt_auto_logout_info.gif"
            alt="자동 로그아웃 안내" id="pop-title" >
        <div id="pop-content">
            <strong>
                자동 로그아웃 남은시간:
                <em id="cntMsg">60</em>초>
            </strong>
            <p>
                고객님의 안전한 금융거래를 위해 로그인 후
                약 10분 동안 서비스 이용이 없어 로그아웃 됩니다.
            </p>
            <p>로그인 시간을 연장하시겠습니까?</p>
        </div>
        <button type="button" class="extend">연장하기</button>
        <button type="button">로그아웃</button>
    </div>
</div>
```

이후, 이 팝업이 사용자에게 노출 될 때, 이 레이어가 자바스크립트에 의해 포커스를 얻을 수 있도록 tabindex를 -1로 설정하고, 사용자가 버튼을 바로 찾을 수 있도록 연장하기 버튼에 포커스를 주도록 한다.

이를 위한 자바스크립트 코드는 다음과 같다.

```
$('#popContent')
    .show()
    .attr({'tabindex' : '-1'})
    .find(' button.extend').focus();
```

그리고, 사용자가 탭키로 레이어를 빠져나가는 것을 방지하기 탭키, Shift + 탭키로 포커스 이동하는 것을 레이어 내부로 가두어지도록 한다.

```
$('#popContent')
    .show()
    .attr({'tabindex' : '-1'})
    .find(' button.extend').focus()
    .on({
        'keydown.focusLock' : focuslockKeyDown,
        'keyup.focusLock' : focuslockKeyUp
    });
...
var shiftPressed = false,
    popup = $('#popContent').get(0),
    firstBtn = $('#popContent button:first').get(0),
    lastBtn =  $('#popContent button:last').get(0);
/**
 * detect shift key released
 * @function focuslockKeyUp
 * @param {event} event
 */
function focuslockKeyUp(event){
    event = event || window.event;
    var keycode = event.which || event.keyCode;
    if( event.shiftKey ){
        shiftPressed = false;
    }
}
/**
 * detect (shift +) tab key pressed
 * @function focuslockKeyDown
 * @param {event} event
 */
function focuslockKeyDown(event){
    event = event || window.event;
    var keycode = event.which || event.keyCode;
    if(event.shiftKey){
        shiftPressed = true;
    }
    if(shiftPressed && keycode === 9 && event.target === popup){
        // 팝업 레이어에서 shift + 탭키 이동
        event.preventDefault ? event.preventDefault() : event.returnValue =
false;
        lastBtn.focus();
    }else if(shiftPressed && keycode === 9 && event.target === firstBtn){
```

```
        // 연장하기 버튼(레이어 내 첫번째 focusable element)에서
        // shift + 탭키 이동
        event.preventDefault ? event.preventDefault() : event.returnValue =
false;
        popup.focus();
    }else if(!shiftPressed &&  keycode === 9 && event.target === lastBtn){
        // 로그아웃 버튼(레이어 내 마지막 focusable element)에서 탭키 이동
        event.preventDefault ? event.preventDefault() : event.returnValue =
false;
        popup.focus();
    }
}
```

그리고 추가적으로 좀 더 접근성을 향상 시키기 위한 방법을 고민해 보자면, 핫키(hotkey) 등을 이용하여 시간을 연장 할 수 있는 기능을 부여하고, 페이지에 접근 했을 때 미리 안내해 주는 방법도 고려해 볼 수 있다.

```
<body>
    <p class="hidden-accessible">
        로그인 후 10분 동안 화면 이동이 없을 시 자동으로 로그아웃 되며,
        로그아웃 1분전에 로그인 연장 화면으로 전환됩니다. <br>
        로그인 연장을 하시려면 키보드의 backspace 키를 누르시면,
        사이트 접속시간이 연장됩니다.
    </p>
    ...
$(document).on({
    'keydown.timeExtend' : function(event){
        event = event || window.event;
        var keycode = event.which || event.keyCode;
        if( keycode === 8 ){
            timeExtend();
        }
    }
});
```

이런 식으로 사용자가 페이지에 접근 시 처음부터 타임 세션의 존재에 대해 인지하게 하고, 해당 대화상자가 열렸을 때 즉각적으로 대응할 수 있는 방법을 제공한다면 좀 더 좋은 사용성을 기대해 볼 수 있을 것으로 예상된다.

14. 드랍다운 메뉴(DropDown Menu)

• 기존 코드의 문제점들

많은 사이트에서 드랍다운 메뉴를 구현할 때 마우스로만 사용이 가능하도록 제공하는 경우가 있다. 이는 마우스를 사용할 수 없는 환경이나 양팔을 사용할 수 없는 상지장애인 등이 큰 불편을 겪을 수 있기 때문에 마우스 이외에 키보드 및 기타 다른 방법으로 해당 메뉴를 사용할 수 있도록 접근성을 준수하려는 노력이 필요하다.

마우스 접근만 허용하고 키보드 접근이 안 되는 경우

또한 마우스와 키보드 접근은 가능하지만 키보드 사용시 원하는 메뉴 선택을 위해 많은 탭키를 사용하는 경우가 있다. 이는 접근성은 보장하고 있으나 사용성 측면에서 다소 불편하다는 단점을 안고 있다.

마우스와 키보드 접근은 가능하나 사용성이 불편한 경우

이런 문제점을 해결하기 위해 우선 마우스 이외에 키보드 사용으로 메뉴선택이 가능하도록 개선하고 키

보드 사용시 상위메뉴 이동은 방향키로, 하위메뉴 이동은 탭키로 접근할 수 있도록 키보드 인터랙션을 개선한다면 접근성과 사용성이라는 두 마리 토끼를 모두 잡을 수 있을 것이다.

키보드 인터랙션 추가를 통해 접근성과 사용성을 높인 경우

• WAI-ARIA를 사용해야 하는 이유

그렇다면 마우스와 키보드 접근성을 보장하고 사용성을 개선하는 방법만이 최선일까? 문제는 아직 남아있다. 아래 코드를 살펴보자.

```
<ul class="mainmenu">
    <li>
        <a href="#academy">아카데미</a>
        <ul id="academy">
            <li><a href="#info">아카데미소개</a></li>
            <li><a href="#history">아카데미연혁</a></li>
            <li><a href="#manager">매니저소개</a></li>
            <li><a href="#map">오시는길</a></li>
        </ul>
    </li>
    <li>
        <a href="#jobeducation">취업과정</a>
        <ul id="jobeducation">
            <li><a href="#strategy">국가기간전략산업</a></li>
            <li><a href="#card">계좌제</a></li>
```

```
            <li><a href="#core">핵심직무과정</a></li>
            <li><a href="#jobs">취업아카데미</a></li>
        </ul>
    </li>
    <li>
        <a href="#story">아카데미 Story</a>
        <ul id="story">
            <li><a href="#techstory">기술스토리</a></li>
            <li><a href="#bookstory">책스토리</a></li>
            <li><a href="#studentstory">연수생스토리</a></li>
        </ul>
    </li>
    <li>
        <a href="#customer">고객센터</a>
        <ul id="customer">
            <li><a href="#notice">공지사항</a></li>
            <li><a href="#tutor">강사모집</a></li>
            <li><a href="#online">온라인결제</a></li>
        </ul>
    </li>
</ul>
```

메뉴 구조를 위해 〈ul〉 요소를 사용한 것을 알 수 있다. 이렇게 〈ul〉 요소로 마크업 한 경우 스크린리더를 사용하는 경우 단순히 목록 시작으로만 읽어주게 된다. 이 때문에 해당 영역이 메뉴인지 일반 목록인지 구분하기 어렵다.

개선 전 스크린리더(NVDA) 음성

- 아카데미 목록 시작
- 아카데미 소개
- 아카데미 연혁
- 매니저 소개
- 오시는 길 목록 끝

이러한 문제를 해결하기 위해 WAI-ARIA를 사용한다면 다음과 같이 스크린리더에서 음성출력을 개선할 수 있다.

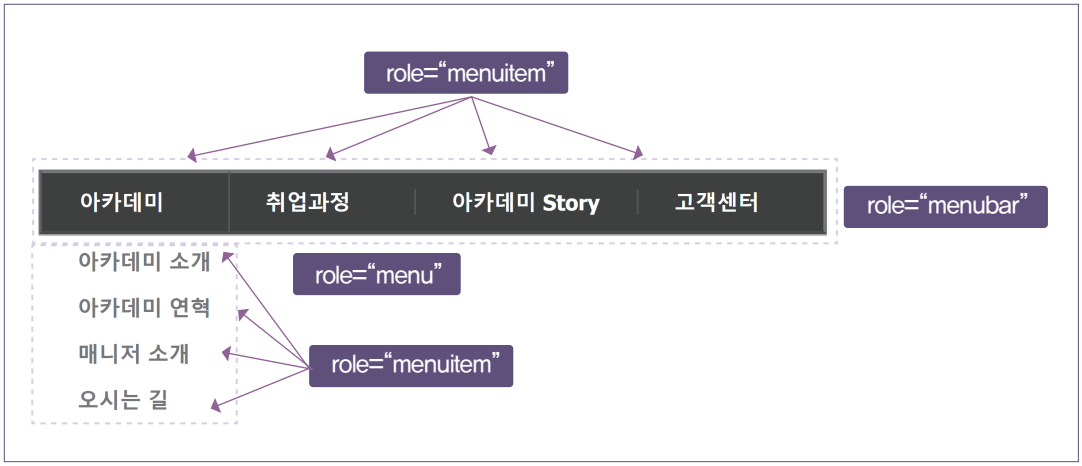
개선 후 스크린리더(NVDA) 음성

- 아카데미 메뉴 시작
- 아카데미 소개
- 아카데미 연혁
- 매니저 소개
- 오시는 길 메뉴 끝

• WAI-ARIA 전체적인 그림과 설명

WAI-ARIA의 역할(Role), 속성(Property), 상태(State)를 이용하여 스크린리더 사용자로 하여금 해당 UI를 인식할 수 있도록 하기위한 구조는 다음과 같다.

드랍다운에 대한 WAI-ARIA 구조



전체를 감싸고 있는 〈ul〉 요소에는 role="menubar"를 설정하고 menubar의 메뉴 아이템인 〈li〉 요소에는 role="menuitem"을 설정한다.

그리고 role="menuitem" (아카데미)의 메뉴 아이템을 설정하기 위해서 자식 요소의 〈ul〉 요소에 role="menu"와 〈li〉 요소에는 role="menuitem"을 설정하여 구성한다.

하위 메뉴중 아카데미 메뉴 외에 tabindex="-1"을 적용한 이유는 아카데미 메뉴 이외에 다른 하위 메뉴에 포커스 진입을 막기 위함이다.

```
<ul class="nav level-1 list-reset" role="menubar" aria-hidden="false">
    <li class="has-subnav" role="menuitem" aria-haspopup="true">
        <a href="#academy">아카데미</a>
        <ul class="level-2 list-reset" data-test="true"
            aria-hidden="true" role="menu">
            <li role="menuitem">
                <a href="#info" tabindex="-1">아카데미소개</a>
            </li>
            <li role="menuitem">
                <a href="#history "tabindex="-1">아카데미연혁</a>
            </li>
            <li role="menuitem">
                <a href="#manager" tabindex="-1">매니저소개</a>
            </li>
            <li role="menuitem">
                <a href="#map" tabindex="-1">오시는길</a>
            </li>
        </ul>
    </li>
</ul>
```

• WAI-ARIA 속성 정리

각 요소에 적용된 역할(Role)과 속성(Property), 상태(State)를 정리해보면 다음과 같다.

요소	역할	속성/상태	설명
〈nav〉	navigation	aria-describedby	role="navigation"이라고 역할을 부여하면 스크린리더에서 "내비게이션 랜드마크"라고 읽는다.
		aria-label	aria-label 속성 값은 화면에 나타나지 않지만 추가적인 설명을 부여할 때 사용할 수 있다. 만약 〈div role="navigation" aria-label="메인메뉴"〉라고 마크업 할 경우 스크린리더는 "내비게이션 랜드마크 메인메뉴"라고 읽는다
〈ul〉	menubar		role="menubar"는 자식 요소를 가질수 있는 상위 메뉴라는 것을 보조기기에게 알려준다.
		aria-hidden	true, false 값을 가질 수 있으며, true일 경우 보조기기에서 들리지 않게 하며, false인 경우 보조기기에서 들리게 한다.(시각적으로 표시되는 것과 무관함)
〈li〉	menuitem		role="menuitem"은 메뉴에 포함된 자식 요소를 구성할 때 사용한다.
		aria-haspopup	true, false 값을 가질 수 있으며, 서브 메뉴의 표시를 알려준다. 값이 true 일 때 스크린리더인 NVDA에서는 "X submenu"로 , JAWS에서는 "X has popup"이라고 읽는다.
〈ul〉	menu		role="menu"는 자식 요소를 가질수 있는 상위 메뉴라는 것을 보조기기에게 알려준다

• 소스 코드 정리

WAI-ARIA를 이용하여 드랍다운을 구현한 소스는 아래와 같다.

```
<!DOCTYPE html>
<html lang="ko">
    <head>
        <meta charset="utf-8">
        <title>ARIA Drop Down Menu</title>
        <style type="text/css">
        .nav { float: left; margin: 20px 0; }
        .nav ul {
            position: absolute;
            top: 2.5em;
            left: -9999px;
            opacity: 0;
            transition: 0.1s linear opacity;
            min-width: 150px;
        }
        .nav li { float: left; position: relative;
        .nav li > a {
            float: left;
            padding: 10px 15px;
            text-decoration: none;
        }
        .nav li > a:hover,
        .nav li > a:focus,
        .nav li:focus > a,
        .nav li:hover > a {
            background: #EFEFEF;
            outline: 0;
        }
        .nav li:hover ul,
        .nav li:focus ul,
        ul.show-menu {
            left: 0;
            opacity: 0.99;
        }
        .nav ul li {
            float: none;
            position: static;
```

```
}
.nav ul a {
    float: none;
    display: block;
    font-size: 12px;
    text-shadow: none;
    transition: 0.1s linear all;
}
.nav ul a:hover,
.nav ul a:focus { text-shadow: none; }
.list-reset {
    margin: 0;
    padding: 0;
    list-style: none;
}
</style>
<script src="http:// code.jquery.com/jquery-1.10.1.min.js"></script>
<script src="http:// code.jquery.com/jquery-migrate-1.2.1.min.js">
</script>
<script>
$(document).ready(function() {
  $('.nav').setup_navigation();
  $('body').addClass('js');
  var $menu = $('#menu'),
      $menulink = $('.menu-link'),
      $menuTrigger = $('.has-subnav > a');
      $menulink.click(function(e) {
            e.preventDefault();
            $menulink.toggleClass('active');
            $menu.toggleClass('active');
        });
      $menuTrigger.click(function(e) {
            e.preventDefault();
            var $this = $(this);
            $this.toggleClass('active').next('ul').toggleClass('active');
        });
});
var keyCodeMap = {
    48:"0", 49:"1", 50:"2", 51:"3", 52:"4", 53:"5", 54:"6", 55:"7",
    56:"8", 57:"9", 59:";",65:"a", 66:"b", 67:"c", 68:"d", 69:"e",
    70:"f", 71:"g", 72:"h", 73:"i", 74:"j", 75:"k", 76:"l",
```

```
        77:"m", 78:"n", 79:"o", 80:"p", 81:"q", 82:"r", 83:"s", 84:"t",
        85:"u", 86:"v", 87:"w", 88:"x", 89:"y", 90:"z",
        96:"0", 97:"1", 98:"2", 99:"3", 100:"4", 101:"5", 102:"6",
        103:"7", 104:"8", 105:"9"
    }
    $.fn.setup_navigation = function(settings) {
        settings = jQuery.extend({
            menuHoverClass: 'show-menu',
        }, settings);
        // 현재 객체에는 role를 menubar로
        // li 객체에는 role를 menuitem으로 설정한다.
        $(this).attr('role', 'menubar').find('li').attr('role', 'menuitem');
        var top_level_links = $(this).find('> li > a');
        // 모든 top-level링크는 tabindex="0"를 추가한다.
        // tabindex를 -1로 설정하면 top_level_links는 메뉴가 오픈될 때까지
        // 포커스를 받지 못한다.
        $(top_level_links).next('ul')
            .attr('data-test','true')
            .attr({ 'aria-hidden': 'true', 'role': 'menu' })
            .find('a')
            .attr('tabIndex',-1);
        // top_level_links 하위에 ul이 있다면 부모의 li 객체에
        // aria-haspopup를 true로 설정한다.
        $(top_level_links).each(function(){
            if($(this).next('ul').length > 0)
                $(this).parent('li').attr('aria-haspopup', 'true');
        });
        $(top_level_links).hover(function(){
            $(this).closest('ul')
             // menu가 보이지 않으면 최상위 ul에
             // aria-hidden="false"를 추가한다.
                .attr('aria-hidden', 'false')
                .find('.'+settings.menuHoverClass)
                .attr('aria-hidden', 'true')
                .removeClass(settings.menuHoverClass)
                .find('a')
                .attr('tabIndex',-1);
            $(this).next('ul')
                .attr('aria-hidden', 'false')
                .addClass(settings.menuHoverClass)
                .find('a').attr('tabIndex',0);
```

```
});
$(top_level_links).focus(function(){
    $(this).closest('ul')
           .find('.'+settings.menuHoverClass)
           .attr('aria-hidden', 'true')
           .removeClass(settings.menuHoverClass)
           .find('a')
           .attr('tabIndex',-1);
    $(this).next('ul')
           .attr('aria-hidden', 'false')
           .addClass(settings.menuHoverClass)
           .find('a').attr('tabIndex',0);
});
// 메뉴 내비게이션을 위한 방향키를 위한 이벤트 처리
$(top_level_links).keydown(function(e){
    if(e.keyCode == 37) {// 왼쪽 방향키
        e.preventDefault();
        // 첫번째 아이템인 경우
        if($(this).parent('li').prev('li').length == 0) {
            $(this).parents('ul')
                   .find('> li').last().find('a').first().focus();
        } else {
            $(this).parent('li').prev('li').find('a').first().focus();
        }
    } else if(e.keyCode == 38) {// 위쪽 방향키
        e.preventDefault();
        if($(this).parent('li').find('ul').length > 0) {
            $(this).parent('li').find('ul')
                   .attr('aria-hidden', 'false')
                   .addClass(settings.menuHoverClass)
                   .find('a').attr('tabIndex',0)
                   .last().focus();
        }
    } else if(e.keyCode == 39) {// 오른쪽 방향키
        e.preventDefault();
        // 마지막 아이템인 경우
        if($(this).parent('li').next('li').length == 0) {
            $(this).parents('ul')
                   .find('> li').first().find('a').first().focus();
        } else{
            $(this).parent('li').next('li').find('a').first().focus();
```

```
                    }
                } else if(e.keyCode == 40) {// 아래쪽 방향키
                    e.preventDefault();
                    if($(this).parent('li').find('ul').length > 0) {
                        $(this).parent('li').find('ul')
                                .attr('aria-hidden', 'false')
                                .addClass(settings.menuHoverClass)
                                .find('a').attr('tabIndex',0)
                                .first().focus();
                    }
                } else if(e.keyCode == 13 || e.keyCode == 32) {// 엔터키 경우
                    // 서브메뉴가 hidden인 경우 서브메뉴를 오픈해라
                    e.preventDefault();
                    $(this).parent('li').find('ul[aria-hidden=true]')
                           .attr('aria-hidden', 'false')
                           .addClass(settings.menuHoverClass)
                           .find('a').attr('tabIndex',0)
                           .first().focus();
                } else if(e.keyCode == 27) {// Esc키일 경우
                    e.preventDefault();
                    $('.'+settings.menuHoverClass)
                        .attr('aria-hidden', 'true')
                        .removeClass(settings.menuHoverClass)
                        .find('a')
                        .attr('tabIndex',-1);
                } else {
                  $(this).parent('li').find('ul[aria-hidden=false]a')
                         .each(function(){
                                if($(this).text().substring(0,1).toLowerCase() ==
keyCodeMap[e.keyCode]) {
                                    $(this).focus();
                                    return false;
                                }
                        });
                }
            });
            var links = $(top_level_links).parent('li').find('ul').find('a');
            $(links).keydown(function(e){
                if(e.keyCode == 38) {// 위쪽 방향키
                    e.preventDefault();
```

```
                    // 첫번째 아이템인 경우
                    if($(this).parent('li').prev('li').length == 0) {
                        $(this).parents('ul').parents('li')
                                .find('a').first().focus();
                    } else {
                        $(this).parent('li').prev('li').find('a').first().focus();
                    }
                } else if(e.keyCode == 40) {// 아래쪽 방향키
                    e.preventDefault();
                    if($(this).parent('li').next('li').length == 0) {
                        $(this).parents('ul').parents('li').find('a').first().focus();
                    } else {
                        $(this).parent('li').next('li').find('a').first().focus();
                    }
                } else if(e.keyCode == 27 || e.keyCode == 37) {
                // Esc 또는 왼쪽 방향키
                    e.preventDefault();
                    $(this)
                        .parents('ul').first()
                        .prev('a').focus()
                        .parents('ul').first()
                        .find('.'+settings.menuHoverClass)
                        .attr('aria-hidden', 'true')
                        .removeClass(settings.menuHoverClass)
                        .find('a')
                        .attr('tabIndex',-1);
                } else if(e.keyCode == 32) {// 스페이스바 키
                    e.preventDefault();
                    window.location = $(this).attr('href');
                } else {
                    var found = false;
                    $(this).parent('li').nextAll('li').find('a').each(function(){
                        if($(this).text().substring(0,1).toLowerCase() ==
keyCodeMap[e.keyCode]) {
                            $(this).focus();
                            found = true;
                            return false;
                        }
                    });
                    if(!found) {
                        $(this).parent('li').prevAll('li').find('a')
                                .each(function(){
```

```
                            if($(this).text().substring(0,1).toLowerCase() ==
keyCodeMap[e.keyCode]) {
                                $(this).focus();
                                return false;
                            }
                        });
                    }
                }
            });
            // 만약 클릭 또는 포커스 이벤트가 내비게이션 밖에서 발생할 경우
            // 메뉴를 감춘다
                $(this).find('a').last().keydown(function(e){
                  if(e.keyCode == 9) {// tab 키
                  // If the user tabs out of the navigation hide all menus
                  $('.'+settings.menuHoverClass)
                      .attr('aria-hidden', 'true')
                      .removeClass(settings.menuHoverClass)
                      .find('a')
                      .attr('tabIndex',-1);
                }
            });
            $(document).click(function(){
                $('.'+settings.menuHoverClass).attr('aria-hidden', 'true')
                    .removeClass(settings.menuHoverClass).find('a')
                    .attr('tabIndex',-1); });

            $(this).click(function(e){
                e.stopPropagation();
            });
        }
        </script>
    </head>
    <body>
    <!-- Navigation -->
            <nav class="menu" id="a11y-menu" role="navigation"
                aria-label="Main menu">
                <ul class="nav level-1 list-reset" role="menubar"
                    aria-hidden="false">
                    <li class="has-subnav" role="menuitem" aria-haspopup="true">
                    <a href="#academy">아카데미</a>
                    <ul class="level-2 list-reset" data-test="true"
                        aria-hidden="true" role="menu">
```

```
            <li role="menuitem"><a href="#info"
                tabindex="-1">아카데미소개</a></li>
            <li role="menuitem"><a href="#history"
                tabindex="-1">아카데미연혁</a></li>
            <li role="menuitem"><a href="#manager"
                tabindex="-1">매니저소개</a></li>
            <li role="menuitem"><a href="#map"
                tabindex="-1">오시는길</a></li>
        </ul>
    </li>
    <li role="menuitem" aria-haspopup="true">
        <a href="#academy">취업과정</a>
        <ul class="level-2 list-reset" data-test="true"
            aria-hidden="true" role="menu" class="list-reset">
            <li role="menuitem"><a href="#strategy"
                tabindex="-1">국가기간전략산업</a></li>
            <li role="menuitem"><a href="#card"
                tabindex="-1">계좌제</a></li>
            <li role="menuitem"><a href="#core"
                tabindex="-1">핵심직무과정</a></li>
            <li role="menuitem"><a href="#jobs"
                tabindex="-1">취업아카데미</a></li>
        </ul>
    </li>
    <li role="menuitem" aria-haspopup="true">
        <a href="#story">아카데미Story</a>
        <ul class="level-2 list-reset" data-test="true"
            aria-hidden="true" role="menu" class="list-reset">
            <li role="menuitem"><a href="#techstory"
                tabindex="-1">기술스토리</a></li>
            <li role="menuitem"><a href="#bookstory"
                tabindex="-1">책스토리</a></li>
            <li role="menuitem"><a href="#studentstory"
                tabindex="-1">연수생스토리</a></li>
        </ul>
    </li>
    <li role="menuitem" aria-haspopup="true">
        <a href="#customer">고객센터</a>
        <ul class="level-2 list-reset" data-test="true"
            aria-hidden="true" role="menu" class="list-reset">
            <li role="menuitem"><a href="#notice"
```

```
                    tabindex="-1">공지사항</a></li>
                <li role="menuitem"><a href="#tutor"
                    tabindex="-1">강사모집</a></li>
                <li role="menuitem"><a href="#online"
                    tabindex="-1">온라인결제</a></li>
            </ul>
        </li>
    </ul>
</nav>
<body>
</html>
```

• 키보드 인터랙션

키보드	인터랙션
왼쪽 방향키	이전 메뉴로 이동, 처음 메뉴일 경우는 마지막 메뉴로 이동
오른쪽 방향키	다음 메뉴로 이동, 마지막 메뉴일 경우는 처음 메뉴로 이동
위쪽 방향키	상위 메뉴로 이동한다. 최상위 일 경우는 최하위 메뉴로 이동
아래쪽 방향키	하위 메뉴로 이동한다. 최하위 일 경우는 최상위 메뉴로 이동
탭키	다음 메뉴나 메뉴아이템으로 이동
Esc키	현재 위치에서 최상위 메뉴로 이동

15. 로딩(Loading)

• 기존 코드의 문제점들

로딩(Loading)이란 처리할 데이터나 이미지가 많아서 페이지가 느린 경우, 다음 페이지로 이동할 때 시간이 필요한 페이지에서 로딩(Loading) 중임을 표기하는 것을 말한다.

이런 로딩 처리는 접근성 관점에서 보면 레이어 팝업과 같이 기존의 방법으로 처리하기가 상당히 곤란하다. 다음은 로딩 중임을 표현하기 위한 코드이다.

```
<div id="main">
    <a href="#" id="test">이동</a>
</div>
<div id="loading" class="display-none">
    <h2>Loading...</h2>
    <p>3초후에 해당페이지로 이동합니다.</p>
</div>
```

이동 링크를 클릭 했을 경우 표시 될 로딩 부분을 〈div id="loading"〉 요소에서 처리하고 있다. 물론 로딩 부분은 CSS로 보이지 않게 display : none 처리가 되어 있고 이동 링크를 클릭할 때 display : block 처리 되어 화면 상에 표시되더라도 스크린리더에서는 여전히 음성 출력을 못하는 문제가 있다.

• WAI-ARIA를 사용해야 하는 이유

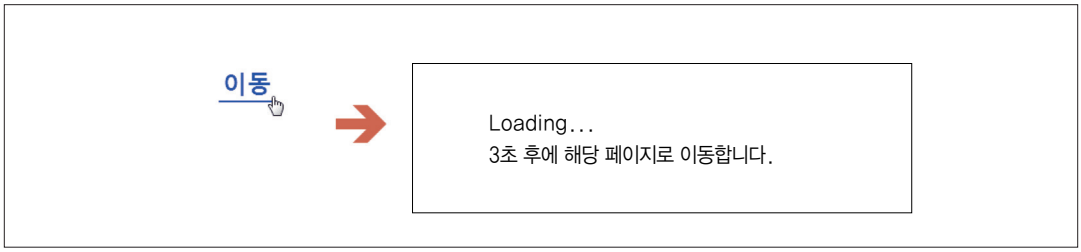
이러한 문제를 개선하기 위해 WAI-ARIA에서는 알림 영역의 실시간 변경 정보를 지정할 수 있도록 role="alert"을 제공하고 있다.

로딩 구현부분인 〈div〉 요소에 role="alert"을 지정하면 스크린리더는 해당 영역에 변경이 일어날 때 변경된 콘텐츠를 읽어주게 된다. 이때 role="alert" 자체가 Live Region이기 때문에 aria-live 속성을 따로 제공하지 않아도 된다.

```
<div id="main">
    <a href="#" id="test">이동</a>
</div>
<div id="loading" role="alert">
 . . .
</div>
```

이렇게 WAI-ARIA를 사용하여 코드를 개선한 경우 스크린리더에서는 이동 링크 클릭 시 나타나는 로딩 화면을 음성으로 출력하는 것을 확인할 수 있다.

WAI-ARIA 속성을 사용하여 실시간으로 로딩 영역의 메시지를 제공한 경우



스크린리더 음성(NVDA)

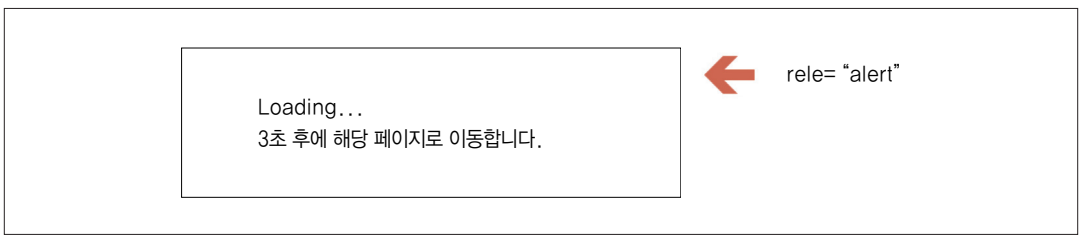
이동 링크

Loading··· , 3초 후에 해당 페이지로 이동합니다.

• WAI-ARIA 전체적인 그림과 설명

WAI-ARIA의 역할(Role), 속성(Property), 상태(State)를 이용하여 스크린리더 사용자로 하여금 해당 UI를 인식할 수 있도록 의미를 부여해보자.

로딩 영역에 WAI-ARIA 역할을 지정



가장 먼저 로딩 영역인 〈div〉 요소에 role="alert"을 추가하여 해당 〈div〉 요소가 로딩 메시지를 보여주는 영역이 되도록 한다.

```
<div id="loading" role="alert">

</div>
```

다음으로 이동 링크를 클릭했을 때 role="alert" 영역에 동적으로 텍스트 내용을 추가하기 위해 자바스크립트로 구현한다.

```
$('#loading').append( // role="alert"의 텍스트 내용을 동적으로 추가한다.
    "<h2>Loading...</h2>"+
    "<p>3초후에 해당페이지로 이동합니다.</p>"
);
```

• WAI-ARIA 속성 정리

각 요소에 적용된 역할(Role)과 속성(Property), 상태(State)를 정리해보면 다음과 같다.

요소	역할	속성/상태	설명
〈div〉	alert		alert 역할을 정의

• 소스 코드 정리

WAI-ARIA를 이용하여 로딩(Loading)을 구현한 소스는 아래와 같다.

```
<!DOCTYPE html>
<html lang="ko">
    <head>
        <meta charset="utf-8">
        <title>Loading(ARIA)</title>
        <link id="size-stylesheet" rel="stylesheet" type="text/css" />
        <script src="http:// code.jquery.com/jquery-1.10.1.min.js"></script>
        <script src="http:// code.jquery.com/jquery-migrate-1.2.1.min.js">
        </script>
        <script type="text/javascript">
            $(document).ready(function(){
                $('#test').click(function(event){
                    event.preventDefault();
                    $('#main').addClass('wrap-back-loading');
                    $('#loading').addClass('wrap-loading');
                    $('#loading').html('');
```

```
            $('#loading').append(
              // aria-live의 내용을 동적으로 추가한다.
                  "<h2>Loading...</h2>"+
                  "<p>3초후에 해당페이지로 이동합니다.</p>"
            );
            setTimeout(hideLoading, 3000);
              // 3초가 경과하면 hideLoading 함수가 실행.
        });
        function hideLoading(){
            location.href = "next.html";
        }
    });
</script>
<style type="text/css" >
    .wrap-back-loading{/*배경 처리*/
         width: 100%;
         height: 100%;
         top: 0px;
         left: 0px;
         position: fixed;
         display: block;
         opacity: 0.2;
         background-color: #626262;
         padding : 10px 10px;
         z-index: 99;
    }
    .wrap-loading{ /*로딩 화면 처리*/
        position:absolute;
        top:50%;
        left:50%;
        width:260px;
        height:120px;
        margin:-70px -140px -70px -140px;
        padding  : 10px 10px;
        background-color: #fff80;
        border : 1px solid black;
    }
    .display-none{ /*감추기*/
        height: 0;
        line-height: 0;
```

```
                overflow: hidden;
                position: absolute;
                text-indent: -9999px;
                width: 0;
            }
        </style>
    </head>
    <body>
        <div id="main"><a href='#' id='test'>클릭</a></div>
        <div id='loading' role='alert'> </div>
    </body>
</html>
```

16. 특수기호

• 기존 코드의 문제점들

HTML 문서를 작성하다 보면 의미를 전달하는 특수기호를 사용해야 하거나 의미를 전달하지 않고 단지 가시적으로 주목을 얻기 위해 특수기호를 사용해야 하는 경우가 종종 있다.

예를 들어, 범위를 의미하기 위한 '~' 이나, 방향의 흐름을 의미하기 위한 '→', 브레드크럼에 상하관계의 표현을 위해 '〉'를 사용하는 경우나, 최근에는 아이콘 웹 폰트를 이용하여 특수 기호를 사용하는 경우도 많아지고 있다.

가격 범위를 표현하기 위한 ~ 문자 사용 예

~ 문자 사용을 위한 마크업 코드

```
<span class="price">
    <em><span class="num _price_reload">3,036,280</span>원</em>
     ~
    <strong><span class="num">7,315,200</span>원</strong>
    <a class="btn_compare" href="..." target="_blank">가격비교 602</a>
</span>
```

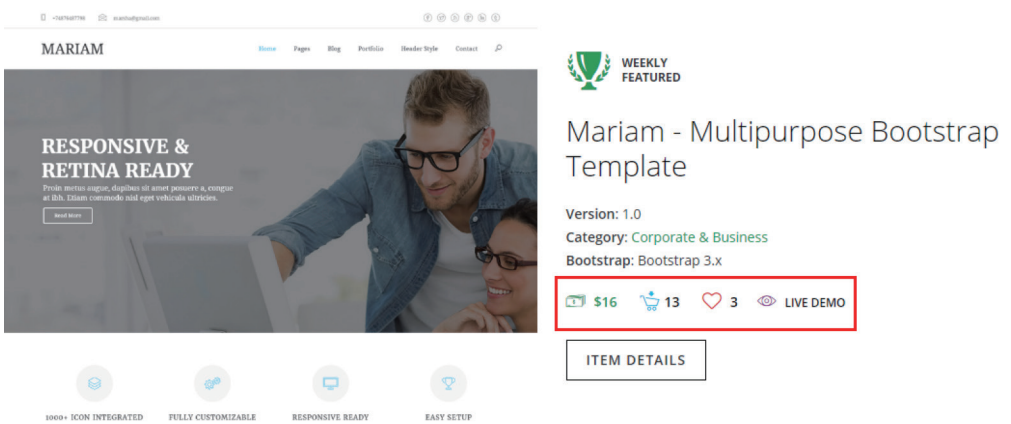
가시적 표현만을 위한 특수기호 사용 예

[YTN 라디오 '최영일의 뉴스. 정면승부']
- 방송 : FM 94.5 (18:10~20:00)
- 방송일 : 2016년 4월 25일 (월요일)
- 대담 : 김성은 뉴스캐스터

가시적 표현만을 위해 사용한 특수기호의 마크업 코드

[YTN 라디오 '최영일의 뉴스. 정면승부']〈br〉
- 방송 : FM 94.5 (18:10~20:00)〈br〉
- 방송일 : 2016년 4월 25일 (월요일)〈br〉
- 대담 : 김성은 뉴스캐스터〈br〉

금액, 구매 횟수, 추천 수 등을 표현하는데 사용된 아이콘 폰트



아이콘 폰트를 사용한 마크업 코드

```
<form action="" class=" purchase " method="post">
    <button class="product-purchase" type="submit">
        <span class="product-price">
            <i class="sb-icon-price"></i>
            <span itemprop="price">$16</span>
        </span>
    </button>
</form>
<form action="" class="cart" enctype="multipart/form-data" method="post">
    <a class="item-add-to-cart" href="#">
        <span class="product-purchased">
            <i class="sb-icon-purchased"></i>12
        </span>
    </a>
</form>
<span class="product-loved">
    <a class="addtofavories" href="#">
        <i class="sb-icon-love"></i> <span>3</span>
    </a>
</span>
<span class="product-preview">
    <a href="..." target="_blank">
        <i class="sb-icon-view"></i> <small>LIVE DEMO</small>
    </a>
</span>
```

위 각 사이트들의 마크업 코드는 좀 더 간결하게 볼 수 있도록 일부 코드는 생략 되었다.

이러한 특수기호의 사용은 스크린리더 사용자에게 있어 많은 혼란을 주기 쉽다.

당장 스크린리더의 설정에 따라 혹은 스크린리더의 인식 가능한 문자 범위에 따라, 특수기호의 읽기가 되는 것이 있고 되지 않은 것이 있으며, 심지어 스크린리더마다 특수기호를 읽어주는 내용이 다르기도 하다.

의미를 전달하지 않아야 하는 특수기호의 경우(예를 들어, 목록의 말머리 기호나 독자의 주목을 끌기 위한 단순한 기호로 사용하는 경우)에 도리어 해당 특수기호를 읽음으로 인해 전달하지 않아야 할 내용까지 전달하게 되는 경우가 발생되기도 한다.

실제로 NVDA 스크린리더의 경우, ■를 “검정 사각형”으로, 〉를 “greater”로 읽어주기 때문에, “가시적 표현만을 위한 특수기호 사용 예”와 같이 마크업 된 인터뷰 전문을 읽을 때에는 오히려 방해가 되는 경우가 된다.

또한, “아이콘 폰트를 사용한 마크업 코드”와 같이 ::before 가상 선택자를 이용하여 SVG로 기호를 그려 넣는 경우에는 오히려 전달해야 할 의미가 있음에도 불구하고 어떠한 내용도 전달하지 못하는 경우 또한 발생된다.

• WAI-ARIA를 사용해야 하는 이유

특수기호는 비 시각장애와 시각장애의 관점에서 두 가지 케이스를 고려해 봐야한다.

의미를 전달하지 않고 단순히 시각적인 효과만을 주는 경우, 스크린리더 사용자에게도 마찬가지로 의미를 전달하지 않도록 되어야 한다.

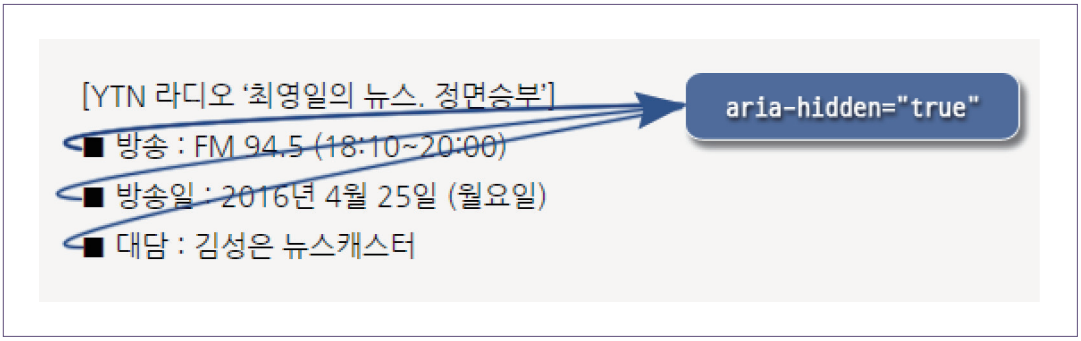
특정한 의미를 전달해야 하는 특수기호의 경우 전달하고자 하는 의미가 명확히 전달 될 수 있도록 해야 한다.

이 두 가지를 만족시키기 위해 WAI-ARIA를 적용하여 비 시각장애인과 시각장애인에게 모두 동등한 수준의 정보를 제공할 수 있도록 구현할 수 있다.

• WAI-ARIA 전체적인 그림과 설명

먼저, 의미가 전달되지 않아야 하는 특수기호의 WAI-ARIA 구조를 그림으로 보도록 하자.

의미가 없는 특수기호 WAI-ARIA 상태(State) 구조



이 경우는 비 스크린리더 사용자에게는 노출이 되어야 하되 스크린리더 사용자에게는 숨겨져야 하는 항목이기 때문에 aria-hidden 속성을 사용하여 스크린리더 사용자에게만 노출되지 않도록 한다.

특정한 의미를 전달하는 특수기호 WAI-ARIA 구조

특수기호를 실제로 전달하고자 하는 의미로 정확하게 전달되도록 하기 위해서는, 스크린리더가 해당 특수기호를 읽지 않도록 설정하고, 실제로 읽어야 할 내용을 읽을 수 있도록 제공이 되어야 한다.

이를 위해 aria-hidden 속성을 사용하고, 스크린리더 사용자에게만 전달할 내용이 추가적으로 제공되어야 한다. 이렇게 aria-hidden 속성이 적용 되었을 때, 각 경우에 대해 스크린리더는 다음과 같은 변화를 보여준다.

의미가 없는 특수기호 스크린리더 음성 출력

WAI-ARIA 적용 이전	WAI-ARIA 적용 이전
NVDA 음성 출력 뷰어 [YTN 라디오 '최영일의 뉴스. 정면승부'] ■ 방송 : FM 94.5 (18:10~20:00) ■ 방송일 : 2016년 4월 25일 (월요일) ■ 대담 : 김성은 뉴스캐스터	NVDA 음성 출력 뷰어 [YTN 라디오 '최영일의 뉴스. 정면승부'] 방송 : FM 94.5 (18:10~20:00) 방송일 : 2016년 4월 25일 (월요일) 대담 : 김성은 뉴스캐스터

특정한 의미를 전달하는 특수기호 스크린리더 음성 출력

WAI-ARIA 적용 이전	WAI-ARIA 적용 이전
NVDA 음성 출력 뷰어 3,036,280원 ~ 7,315,200원 링크 가격비교 602	NVDA 음성 출력 뷰어 3,036,280원 부터 7,315,200원 링크 가격비교 602

이렇게 aria-hidden 속성이 true로 적용된 요소는 가시적으로는 표현이 되지만, 스크린리더는 해당 요소를 숨김으로 처리하여 사용자에게 해당 내용을 전달하지 않게 된다.

따라서 이를 이용하여 스크린리더가 특수기호를 읽지 않도록 설정하고, 스크린리더 사용자를 위한 숨김 텍스트(가시적으로는 숨겨지지만 스크린리더 사용자에게는 읽어줄 수 있는)를 제공하는 방향으로 적용하게 되면 시각장애인과 비 시각장애인 모두에게 동등한 수준의 정보를 전달해 줄 수 있게 된다.

• WAI-ARIA 속성 정리

요소	역할	속성/상태	설명
〈span〉, 〈i〉		aria-hidden	들리지 않아야 하는 경우 true, 들려야 하는 경우 false
button		aria-label	해당 요소의 레이블 설정

• 소스 코드 정리

이제, 앞서 설명한 대로 접근 가능한 특수기호를 만드는 방법을 보도록 하자.
(본 예제에서는 아이콘 폰트의 마크업을 나타내기 위해 Font Awesome의 마크업을 인용했다.)

먼저, 의미를 전달하지 않고 단순히 시각적인 효과만을 나타내는 경우의 코드이다.

단순 시각적 효과만을 나타내는 경우에는 스크린리더 사용자에게 어떠한 정보도 전달되지 않도록 해야 하기 때문에 해당 특수기호에 aria-hidden="true"를 설정해야 한다.

```
<span aria-hidden="true">■</span>  방송 : FM 94.5 (18:10~20:00)<br>
<span aria-hidden="true">■</span> 방송일 : 2016년 4월 25일 (월요일)<br>
<span aria-hidden="true">■</span> 대담 : 김성은 뉴스캐스터<br>
<i class="fa fa-comment" aria-hidden="true" ></i> 댓글
```

다음은 특정한 의미를 전달해야 하는 경우에 대한 예이다.

이러한 경우에는 특수기호 자체는 스크린리더가 읽지 못하도록 aria-hidden="true"를 설정하고, 사용자에게 전달하고자 하는 의미를 숨김 텍스트로 제공하도록 한다.

```
<span class="price">
    <em><span class="num _price_reload">3,036,280</span>원</em>
    <span aria-hidden="true">~</span>
    <span class="hidden-accessible">부터 </span>
    <strong><span class="num">7,315,200</span>원</strong>
    <a class="btn_compare" href="..." target="_blank">가격비교 602</a>
</span>
<p>
    <i class="fa fa-exclamation-triangle" aria-hidden="true"></i>
    <span class="hidden-accessible">경고</span> 삭제 후에는 되돌릴 수 없으니
    신중히 선택하시기 바랍니다.
</p>
```

마지막으로 인터랙션 요소를 나타내는 특수기호 혹은 아이콘 폰트의 경우이다.

이 경우에는 스크린리더 사용자를 위한 숨김 텍스트를 제공하지 않고, 인터랙션 요소에 title 속성(Attiribute)을 주는 것만으로 정보 전달이 이루어질 수 있다.

```
<button type="button" title="닫기">
    <span aria-hidden="true">X</span>
</button>
<button type="button" title="facebook으로 공유하기">
    <i class="fa fa-facebook-square" aria-hidden="true"></i>
</button>
<button type="button" title="twitter로 공유하기">
    <i class="fa fa-twitter-square" aria-hidden="true"></i>
</button>
<button type="button" title="pinterest로 공유하기">
    <i class="fa fa-pinterest" aria-hidden="true"></i>
</button>
```

W3C에서는 이 방법을 권장하고 있으나, 스크린리더의 설정에 따라 title 속성 값을 읽지 않도록 사용하는 경우도 있기 때문에 aria-label 속성을 사용하여 정보를 전달하는 방법도 고려해 봄직 하다.

```
<button type="button" aria-label="facebook으로 공유하기"
        title="facebook으로 공유하기">
    <i class="fa fa-facebook-square" aria-hidden="true"></i>
</button>
<button type="button" aria-label="twitter로 공유하기" title="twitter로 공유하기">
    <i class="fa fa-twitter-square" aria-hidden="true"></i>
</button>
<button type="button" aria-label="pinterest로 공유하기"
        title="pinterest로 공유하기">
    <i class="fa fa-pinterest" aria-hidden="true"></i>
</button>
```

인터랙션 요소에 aria-label 속성(Property) 혹은 title 속성(Attribute)을 적용할 경우, 해당 요소가 특정한 기능과 상태 정보를 동시에 제공하는 경우라면 주의를 기울여야 할 필요가 있다.

물론, aria-pressed 혹은 aria-checked 속성 등의 적절한 상태(state)를 이용하는 것이 더 좋을 것으로 기대되지만, aria-label 속성(Property) 혹은 title 속성(Attribute)으로만 제공해야 하는 경우에는 사용자로 하여금 해당 기능과 상태를 명확하게 인지 할 수 있도록 제공하는 것이 좋다.

만일 스크랩하기 기능과 스크랩 된 상태 정보를 표현하기 위해 다음과 같은 아이콘 폰트를 사용했다고 가정해보자.

기능/상태 정보를 동시에 전달하는 인터랙션 요소의 예

이 경우에는 다음과 같은 마크업이 좀 더 적절할 것으로 기대된다.

```
<button type="button" aria-label="스크랩 하기 " title="스크랩 하기">
    <i class="fa fa-bookmark" aria-hidden="true"></i>
</button> 49
<button type="button" aria-label="스크랩 되어 있음,
        스크랩 해제하기" title="스크랩 해제하기">
    <i class="fa fa-bookmark-o" aria-hidden="true"></i>
</button> 50
```

즉, aria-label 속성의 값에 상태 정보와 기능 정보를 모두 담아, 텍스트 정보만으로 모든 정보를 파악해야 하는 스크린리더 사용자에게 스크랩이 되었는지의 여부 정보와 이 버튼을 눌렀을 때 어떤 동작이 일어날 것인지에 대한 기능 정보가 무엇인지를 명확하게 전달해 주는 것이 더 좋은 정보 전달이 될 것으로 기대할 수 있다.

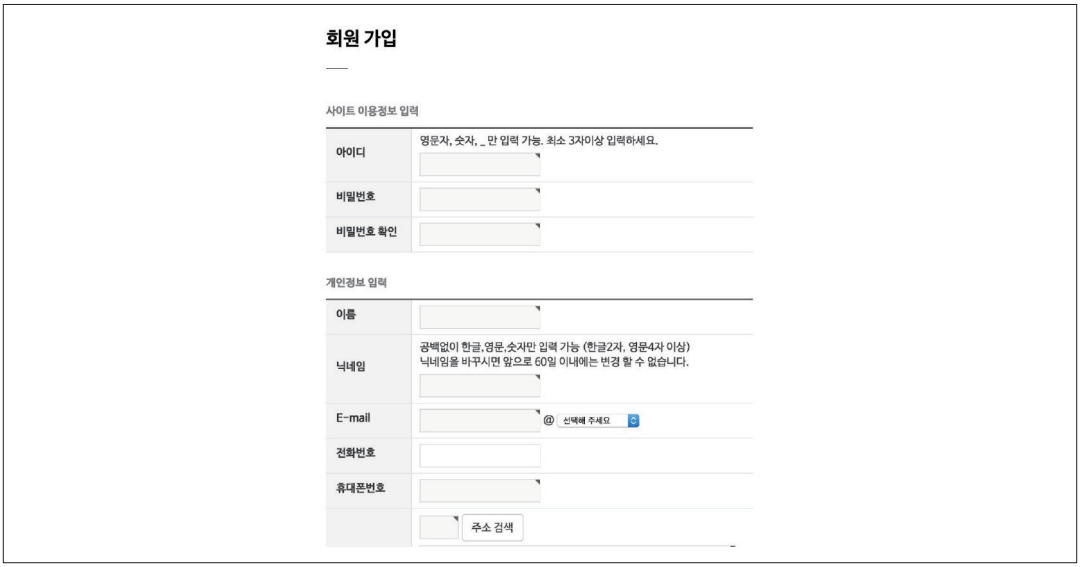
17. 폼 레이블(Form Label)

• 기존 코드의 문제점들

폼(Form)은 사용자와 웹사이트/애플리케이션이 상호작용하는데 사용되는 중요한 기술 중 하나로, 사용자가 입력한 데이터를 웹 서버에 전송할 경우 사용된다.

폼은 텍스트 필드(input, textarea), 셀렉트박스(Selectbox), 버튼(Button), 체크박스(Checkbox), 라디오(Radio) 버튼과 같은 다양한 컴포넌트 조합으로 이루어지는데 대부분 이러한 컴포넌트는 컴포넌트를 설명하는 레이블과 함께 사용된다. 이때 각 컴포넌트에 정확한 레이블을 제공하지 않을 경우, 사용자는 입력에 혼란을 겪을 수 있다.

kitchendogue.com의 회원가입 폼 UI



회원가입 목적을 구현한 폼 UI 디자인이지만, 앞의 회원가입 폼과 달리 다음의 회원가입 폼은 정확한 레이블 및 설명을 제공하지 않아 사용자가 입력하는데 많은 혼란을 야기한다.

dewytree.com 회원가입 폼 UI

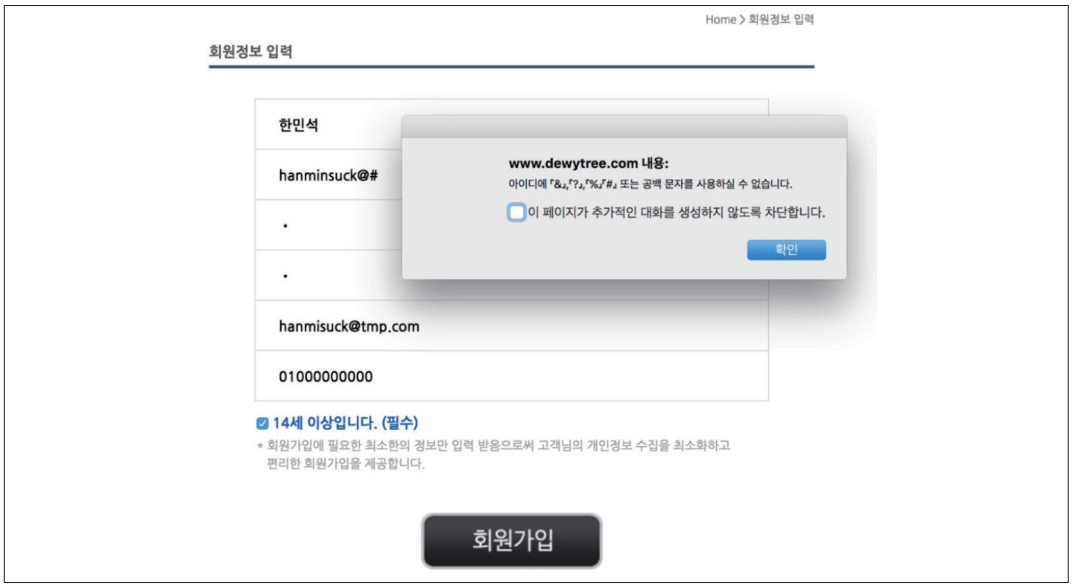
얼핏보면 각 텍스트 입력 필드에 레이블을 제공한 것처럼 생각할 수 있으나, 사용자가 되어 정보를 입력해보면 혼란에 휩싸인다.

특수문자 사용 여부라던가, 입력 가능한 글자 개수, 또는 필수 입력 사항 같은 정보를 사전에 제공해주지 않기 때문이다.

혼란스러움을 뒤로하고 사용자는 회원가입을 위해 주어진 입력 필드를 모두 기입한 후 회원가입 버튼을 눌러보지만, 그제서야 이렇게 입력하면 안된다며 오류메시지 창을 띄워준다.

결국 정확한 정보를 레이블 또는 설명으로 제공하지 않음으로 인해 사용자에게 불편을 끼치고 시간을 낭비하게 만든다.

dewytree.com 회원가입 시 발생한 오류 메시지



이와 같이 레이블을 정확하게 제공하지 않을 경우, 스크린리더 사용자는 더더욱 혼란스러울 수 있다. 사용자 이름을 입력하는 텍스트 필드에 포커스 된 상태를 스크린리더가 읽어보지만 레이블이 제공되지 않아 어떤 정보를 입력해야 하는지 알 수 없다.

이름 입력 필드 포커스 상태를 스크린리더에서 출력

스크린리더가 정확하게 텍스트 입력 필드 레이블을 읽지 못한 이유를 살펴보면 해당 폼은 〈label〉 요소를 제공하고 있으나, 〈input〉 요소에 레이블임을 명확하게 명시하지 않아 스크린리더가 읽어주지 못한다.

```
<li>
  <label>이름</label>
  <input
    type="text"
    name="hname"
    id="hname"
    class="MS_input_txt w137 txt-input input-label"
    size="15"
    maxlength="30"
    style="opacity: 0;">
</li>
```

스크린리더 음성(NVDA)

편집 창

• WAI-ARIA를 사용해야 하는 이유

앞서 다룬 예의 문제를 해결하는데 WAI-ARIA가 반드시 필요한 것은 아니다. 〈label〉 요소에 for 속성을 사용하여 연결할 〈input〉 요소의 id 속성 값과 동일한 값을 입력하면 스크린리더는 정확하게 레이블과 함께 텍스트 입력 필드를 읽어줄 수 있어 사용자에게 정확한 정보를 제공할 수 있다.

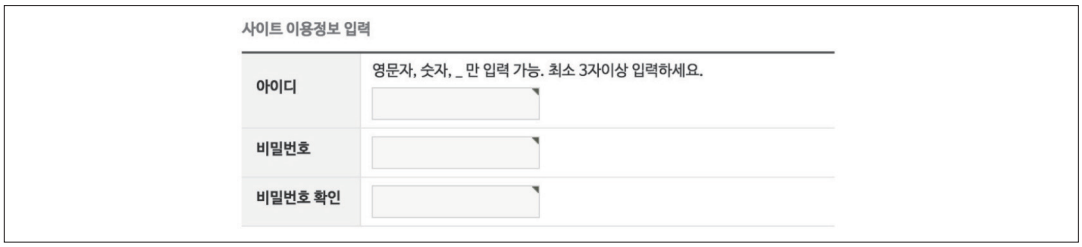
```
<li>
  <label for="hname">이름</label>
  <input
    type="text"
    name="hname"
    id="hname"
    class="MS_input_txt w137 txt-input input-label"
    size="15"
    maxlength="30"
    style="opacity: 0;">
</li>
```

스크린리더 음성(NVDA)

이름 편집 창

하지만 살펴 본 전통적인 레이블 방법으로는 〈input〉 요소에 레이블을 제공할 수는 있으나, 명시적으로 어떤 값을 입력해야 하는지를 기술하는 설명을 제공할 수는 없다. 예를 들어 아래 아이디 입력 필드는 사용자가 입력 가능한 정보를 〈span〉 요소로 제공하고 있으나, 〈label〉 요소처럼 〈input〉 요소와 관계가 명시적으로 연결된 것은 아니다.

아이디 입력 필드 레이블과 설명

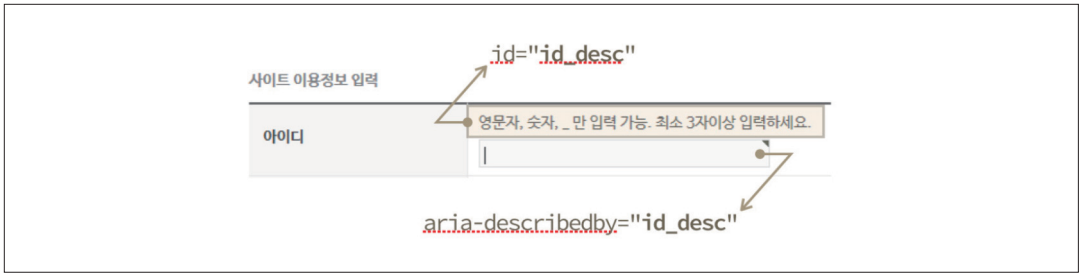


이런 경우 WAI-ARIA를 사용하면 〈span〉 요소와 〈input〉 요소를 관계적으로 연결 가능할 뿐만 아니라, 스크린리더에서 레이블과 함께 설명을 읽어줄 수 있어 접근성을 향상시킬 수 있다.

• WAI-ARIA 전체적인 그림과 설명

아이디 입력 시 주의해야 할 사항(설명)을 〈input〉 요소에 연결하는 것은 매우 쉽다. 〈input〉요소에 aria-describedby 속성을 추가한 다음, 설명 내용을 담고 있는 〈span〉 요소의 id 속성 값과 동일하게 작성해주기만 하면 된다.

아이디 입력 필드에 WAI-ARIA를 사용하여 설명을 연결



이와 같이 간단한 조작만으로도 접근성을 향상시킬 수 있다. WAI-ARIA가 적용된 입력 필드를 스크린리더를 통해 읽어 보면 다음과 같이 읽어준다.

스크린리더 음성(NVDA)

아이디 필수 편집창 영문자, 숫자, _ 만 입력 가능. 최소 3자 이상 입력하세요.

Note.

aria-describedby 속성은 폼 컴포넌트 외에 다른 곳에서도 사용 가능하다.

• WAI-ARIA 속성 정리

앞서 살펴본 예제를 포함해 폼 컴포넌트에 적용 가능한 WAI-ARIA 속성을 정리해보자. 기본적으로는 HTML 레이블 방법을 사용하고, 여기에 WAI-ARIA를 추가 적용하면 보다 향상된 접근성을 제공할 수 있다.

요소	역할	속성/상태	설명
〈label〉			레이블에 연결할 요소의 식별자(id) 설정
〈input〉 〈textarea〉 〈select〉 〈button〉			컴포넌트 식별자(id) 설정
		aria-label=""	레이블이 시각적으로 제공하기 어려울 경우 레이블로 읽힐 수 있도록 설정
		aria-labelledby=""	레이블이 있을 경우 레이블 요소의 id 값 입력
		aria-describedby=""	설명 글이 있을 때 설명 글의 id 값을 입력
		aria-required="true"	필수 입력항목일 경우 true 값을 입력
		aria-invalid="true"	입력 항목이 유효하지 않으면 true 값으로 변경
〈div〉 〈span〉			설명 글 식별자 설정
	alert		입력 항목이 유효하지 않을 경우 화면에 표시되어 사용자에게 입력 값이 잘못 되었음을 알리는 역할 설정

• 소스 코드 정리

살펴 본 WAI-ARIA를 활용하여 폼 컴포넌트의 접근성을 향상시키는 예제를 과정 별로 살펴보자. 먼저 HTML에서 기본 지원하는 레이블 방법을 사용하여 폼 컴포넌트 코드를 작성한다.

```
<div class="form-group">
  <!-- 레이블 -->
  <label for="register_id">아이디</label>
  <!-- 입력 시 알아야 할 정보 -->
  <span id="id_info">영문자, 숫자, _ 만 입력 가능. 최소 3자이상 입력.</span>
  <!-- 사용자 입력 필드 -->
  <input
    type="text"
    name="register_id"
    id="register_id"
    minlength="3"
    maxlength="20"
    required>
</div>
```

이어서 WAI-ARIA 속성 중 aria-describedby 속성을 사용하여 아이디 입력 설명 글을 연결한다. 이를 통해 스크린리더 사용자는 아이디를 기입할 때 정확하게 입력 가능해진다.

```
<div class="form-group">
  <!-- 레이블 -->
  <label for="register_id">아이디</label>
  <!-- 입력 시 알아야 할 정보 -->
  <span id="id_info">영문자, 숫자, _ 만 입력 가능. 최소 3자이상 입력.</span>
  <!-- 사용자 입력 필드 -->
  <input type="text"
    name="register_id"
    id="register_id"
    minlength="3"
    maxlength="20"
    required
    aria-describedby="id_info" >
</div>
```

다음으로 스크린리더 사용자에게 해당 폼 컴포넌트가 필수 입력 사항임을 읽어 알려줄 수 있도록 aria-required 속성 값을 추가한다.

```
<div class="form-group">
  <!-- 레이블 -->
  <label for="register_id">아이디</label>
  <!-- 입력 시 알아야 할 정보 -->
  <span id="id_info">영문자, 숫자, _ 만 입력 가능. 최소 3자이상 입력.</span>
  <!-- 사용자 입력 필드 -->
  <input type="text"
    name="register_id"
    id="register_id"
    minlength="3"
    maxlength="20"
    required
    aria-describedby="id_info"
    aria-required="true" >
</div>
```

HTML 마크업 마무리로 사용자가 현재 입력한 값이 유효한지(valid), 유효하지 않은지(invalid) 상태를 제공할 aria-invalid 속성을 추가한다. 그리고 사용자가 입력한 값이 잘못되었을 경우, 화면에 표시하고 스크린리더 사용자에게 이를 안내하여 읽어줄 수 있도록 alert 역할을 가진 〈span〉 요소를 추가한 후 식별 가능한 id 속성을 추가한다.

```
<div class="form-group">
  <!-- 레이블 -->
  <label for="register_id">아이디</label>
  <!-- 입력 시 알아야 할 정보 -->
  <span id="id_info">영문자, 숫자, _ 만 입력 가능. 최소 3자이상 입력.</span>
  <!-- 사용자 입력 필드 -->
  <input type="text"
    name="register_id"
    id="register_id"
    minlength="3"
    maxlength="20"
    required
    aria-describedby="id_info"
    aria-required="true"
    aria-invalid="false" >
  <!-- 입력 정보가 유효하지 않을 경우, 화면에 표시하고 읽어 줄 주의 정보 -->
  <span role="alert" id="id_error"></span>
</div>
```

자바스크립트에서는 키 입력 값을 체크하는 이벤트를 통해 사용자가 입력한 값을 검증하여 올바르지 않은 값을 입력했을 경우, 사용자에게 주의 내용을 제공하고, 스크린리더가 이를 읽을 수 있도록 상태 변경을 처리한다.

```
// 아이디 입력 필드 참조
var $r_id = $('#register_id');
// 이벤트 핸들러(함수) 연결
$r_id.on('keyup', $.proxy(validateId, $r_id));
// 아이디 유효성 검사 함수
function validateId(e) {
  // 사용자가 입력한 아이디 값 참조(양쪽 공백 제거)
  var value = $.trim( this.val() );
  // 아이디 최소 입력 값
  var min = this.attr('minlength') || 0;
  // 아이디 최대 입력 값
  var max = this.attr('maxlength') || 20;
  // 주의 요소 참조
  var $alert = this.next('[role="alert"]');
  // 정규표현식 객체 생성
  var reg_id = new RegExp('^[A-Za-z0-9_]{'+min+','+max+'}$');
  // 사용자 입력 값 확인 검증
  // 유효하지 않은 값을 입력한 경우 상태 변경
  if ( !reg_id.test(value) ) {
    this.attr({
      'aria-invalid': true,
      'aria-describedby': 'id_error'
    });
    $alert.text('영문자, 숫자, _만 입력 가능(최소 3자)');
}
  // 유효한 값을 입력한 경우 상태 변경
  else {
    this.attr({
      'aria-invalid': false,
      'aria-describedby': 'id_info'
    });
    $alert.text('');
  }
}
```

18. 페이지 내비게이터 (Page Navigator)

• 기존 코드의 문제점들

페이지 내비게이터(Page Navigator)는 웹사이트/웹 애플리케이션의 콘텐츠가 많을 경우, 이를 페이지 간 탐색이 가능하게 구성된 컴포넌트로 다음과 같은 스타일이 일반적이다.

페이지 내비게이터 UI 컴포넌트

페이지 내비게이터 UI 컴포넌트 구현 시, 중요한 키 포인트는 페이지 간 탐색이 가능한 링크 기능을 제공하는 것과 현재 활성화된 페이지 링크를 시각적으로 구분 가능하도록 표현해야 한다.

색을 인지하지 못하는 사용자를 고려하여 색상만으로 구분하는 것이 아닌, 모양으로도 식별 가능하게 구성해야 한다.

「페이지 내비게이터」 HTML 마크업

```
<div class="ui-page-navigator">
    <a id="pgn-a-prev" href="/" title="이전 페이지" class="ui-pgn-prev">
        <em class="readable-hidden">이전 페이지</em>
    </a>
    <a id="pgn-a-1" href="/"><span>1</span></a>
    <a id="pgn-a-2" href="/"><span>2</span></a>
    <a id="pgn-a-3" href="/"><span>3</span></a>
    <a id="pgn-a-4" href="/"><span class="ui-pgn-active">4</span></a>
    <a id="pgn-a-5" href="/"><span>5</span></a>
    <a id="pgn-a-6" href="/"><span>6</span></a>
    <a id="pgn-a-7" href="/"><span>7</span></a>
    <a id="pgn-a-8" href="/"><span>8</span></a>
    <a id="pgn-a-9" href="/"><span>9</span></a>
    <a id="pgn-a-10" href="/"><span>10</span></a>
    <a id="pgn-next" href="/" title="다음 페이지" class="ui-pgn-next">
        <em class="readable-hidden">다음 페이지</em>
    </a>
</div>
```

뿐만 아니라 스크린리더 사용자가 현재 페이지의 위치를 알 수 있도록 현재 페이지 색인을 제공해야 한다. 단, "현재 페이지"라는 텍스트가 화면에 노출되지 않고 스크린리더에서는 읽을 수 있도록 숨김처리 한다.

「현재 활성화된 페이지」 HTML 마크업

```
<a id="pgn-anchor-4" href="/">
    <span class="readable-hidden">현재 페이지</span>
    <span class="ui-pgn-active">4</span>
</a>
```

기술한 방법은 페이지 링크 간 탐색이 가능할 뿐 아니라, 스크린리더 사용자에게 현재 활성화된 페이지 색인을 잘 알려준다.

스크린리더 음성(NVDA)

현재 페이지 4 링크

• WAI-ARIA를 사용해야 하는 이유

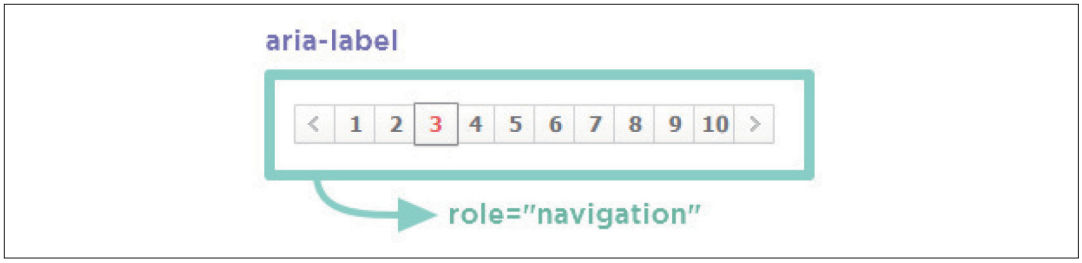
앞서 구현된 페이지 내비게이터 UI 컴포넌트에 큰 문제는 없지만, 내비게이션 역할을 수행 함에도 단순하게 링크를 〈div〉 요소로 그룹화하여 제공하는 것보다 내비게이션(navigation) 역할을 부여하는 것이 보다 의미적이며 유효 적절할 것이다.

뿐만 아니라 현재 탐색 중인 영역이 페이지 내비게이터 UI 컴포넌트 임을 스크린리더 사용자에게 알려줄 수 있는 레이블(Label)을 제공할 필요가 있다.

• WAI-ARIA 전체적인 그림과 설명

페이지 내비게이터 UI 컴포넌트에 WAI-ARIA를 적용하는 방법은 내비게이션 역할을 설정하기 위한 role과 레이블을 제공하기 위한 aria-label 속성을 사용한다.

페이지 내비게이터 UI 컴포넌트에 설정된 WAI-ARIA 구조



• WAI-ARIA 속성 정리

앞서 살펴본 페이지 내비게이터 UI 컴포넌트에 적용한 WAI-ARIA 속성을 정리해보자.

요소	역할	속성/상태	설명
〈div〉	navigation		내비게이션 역할 설정
		aria-label	내비게이션 레이블 설정

• 소스 코드 정리

페이지 내비게이터 UI 컴포넌트 접근성 향상을 위한 WAI-ARIA 속성을 예제에 추가해보자. 앞서도 이야기 했지만, 적용 방법은 매우 쉽고 간단하다.

다음과 같이 〈div〉 요소로 그룹화 한 페이지 내비게이터 영역에 WAI-ARIA 속성을 추가해주면 된다.

```
<div role="navigation" aria-label="페이지 선택" class="ui-page-navigator">
    <a id="pgn-a-prev" href="/" title="이전 페이지" class="ui-pgn-prev">
        <em class="readable-hidden">이전 페이지</em>
    </a>
    <a id="pgn-a-1" href="/"><span>1</span></a>
    <a id="pgn-a-2" href="/"><span>2</span></a>
    <a id="pgn-a-3" href="/"><span>3</span></a>
    <a id="pgn-a-4" href="/">
        <span class="readable-hidden">현재 페이지</span>
        <span class="ui-pgn-active">4</span>
    </a>
    <a id="pgn-a-5" href="/"><span>5</span></a>
    <a id="pgn-a-6" href="/"><span>6</span></a>
```

```
    <a id="pgn-a-7" href="/"><span>7</span></a>
    <a id="pgn-a-8" href="/"><span>8</span></a>
    <a id="pgn-a-9" href="/"><span>9</span></a>
    <a id="pgn-a-10" href="/"><span>10</span></a>
    <a id="pgn-next" href="/" title="다음 페이지" class="ui-pgn-next">
        <em class="readable-hidden">다음 페이지</em>
    </a>
</div>
```

19. 실시간 폼 피드백

• 기존 코드의 문제점들

과거의 전통적인 웹 에서는 사용자가 전송 버튼을 클릭하여 데이터 전송을 일으키는 시점에 양식 내 각 항목들에 대한 유효성 검사를 진행하여 잘못 입력 된 부분을 경고창(alert)을 이용하여 피드백을 주는 형태가 많이 사용되어 왔으나, 현재 많은 웹 사이트들에서는 각 입력 항목에 대해 실시간으로 검증된 결과에 대한 피드백을 주는 것을 쉽게 볼 수 있다.

실시간으로 입력 값을 검증하는 회원가입 양식

위 그림과 같이 이미 가입된 아이디를 넣었을 경우, 해당 입력 상자 근처에 이 입력 값에 대한 유효성 검사 결과를 노출시키고, 비밀번호에 대한 유효성 검사의 결과를 노출시켜, 사용자로 하여금 즉시 피드백을 받을 수 있어 편리한 UI(User Interface)로 많은 곳에 사용되고 있는 것을 볼 수 있다.

그런데, 과연 이러한 UI(User Interface)가 접근성 면에서는 어떠한지 살펴보아야 할 필요가 있는데, 위 사이트의 코드를 살펴보면 다음과 같이 마크업 되어 있다.

```
<form action="..." id="joinForm" method="post" name="joinForm">
    <div class="join-email">
        <div class="join-row">
            <span class="join-email-title">아이디(이메일)</span>
        </div>
        <div class="join-row">
            <input autocomplete="off" type="text">
            <span class="join-at">@</span>
            <input autocomplete="off" type="text">
            <select class="join-email-select-domain"> ... </select>
        </div>
        <div class="form-err join-email-errmsg">
            이미 가입된 이메일 주소입니다. 다른 이메일을 입력하여 주세요.
            <a href="/login/login.pang">로그인</a>
            <a href="/login/findPassword.pang">비밀번호 찾기</a>
        </div>
    </div>
    <div class="join-password">
        <div class="join-password-col">
            <span class="join-password-title">비밀번호</span>
            <input autocomplete="off" class="..." id="join-password-input1"
                   type="password">
        </div>
        <div class="join-password-col">
            <span class="join-password-title">비밀번호 확인</span>
            <input autocomplete="off" class="..." id="join-password-input2"
                   type="password">
        </div>
    </div>
    <div class="join-password-notice">
        비밀번호는 6~15자 이내로 ...<br>
        단, 3가지 모두를 조합하실 경우 더욱 강력한 패스워드 구현이 가능합니다.
    </div>
    <div class="form-err join-password-errmsg">
        비밀번호는 영자로만 입력할 수 없습니다.
    </div>
    ...
</form>
```

위 코드에서 하이라이트 된 부분이 입력 값에 따라 초기에는 비어 있다가 동적으로 추가된 피드백 부분이다.

비 스크린리더 사용자의 경우에는 실시간으로 반영되는 피드백을 시각 정보로 바로 받아 들일 수 있으므로 문제가 되지 않겠지만, 스크린리더 사용자에게는 해당 피드백이 추가되는 시점에 해당 텍스트를 인지하기란 어려운 일이 된다.

물론, 하나의 입력 상자에 값을 입력 한 후 브라우저 모드로 이후 내용을 탐색하게 되면 해당 텍스트를 읽을 수 있겠지만, 만일 각 입력 상자를 탭키로 이동 하여 사용하는 경우에는 해당 텍스트를 뛰어 넘어 가기 때문에 피드백을 전달 받을 수 없는 상황에 이르게 되기 마련이고, 사용자로 하여금 피드백을 확인하게 하기 위해 브라우저 모드와 포커스 모드 사이를 계속해서 스위칭하여 탐색하도록 하는 것은 매우 불편을 만드는 일임에는 틀림이 없을 것이다

• WAI-ARIA를 사용해야 하는 이유

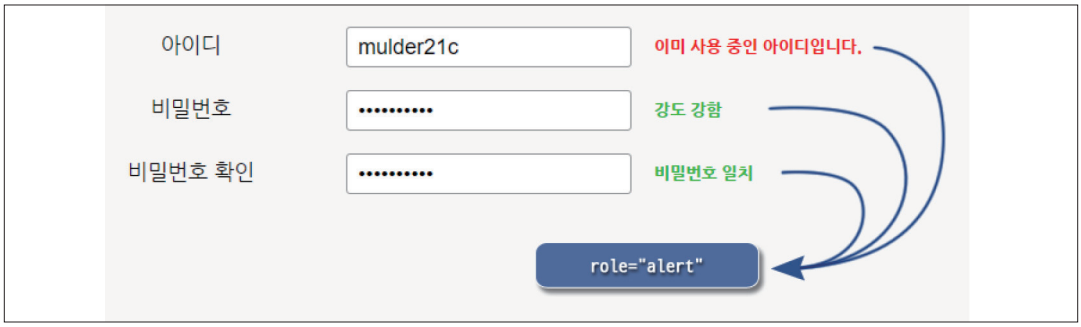
위와 같은 경우, WAI-ARIA의 Live Region을 사용한다면, 변경되는 문서의 내용을 스크린리더 사용자에게 실시간으로 전달 할 수 있기 때문에 비단 비 스크린리더 사용자에게뿐 아니라 스크린리더 사용자에게 까지도 전달하고자 하는 정보를 동일한 시점에 동일하게 전달 할 수 있는 효과를 가질 수 있다.

그에 따라 스크린리더 사용자는 자신의 입력 값에 대한 피드백을 찾기 위해 브라우저모드 - 포커스 모드 사이를 계속 왔다 갔다 해야 하는 불편함이 사라지고 실시간으로 피드백을 받기 때문에 좀 더 빠르게 대응할 수 있게 된다.

• WAI-ARIA 전체적인 그림과 설명

전체적인 WAI-ARIA의 역할(Role) 구조를 그림으로 살펴보자.

WAI-ARIA 역할구조

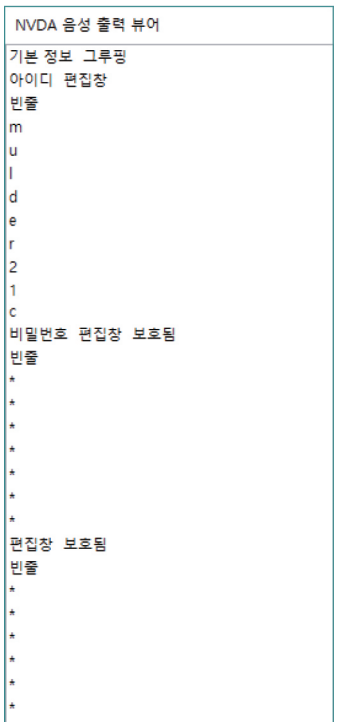
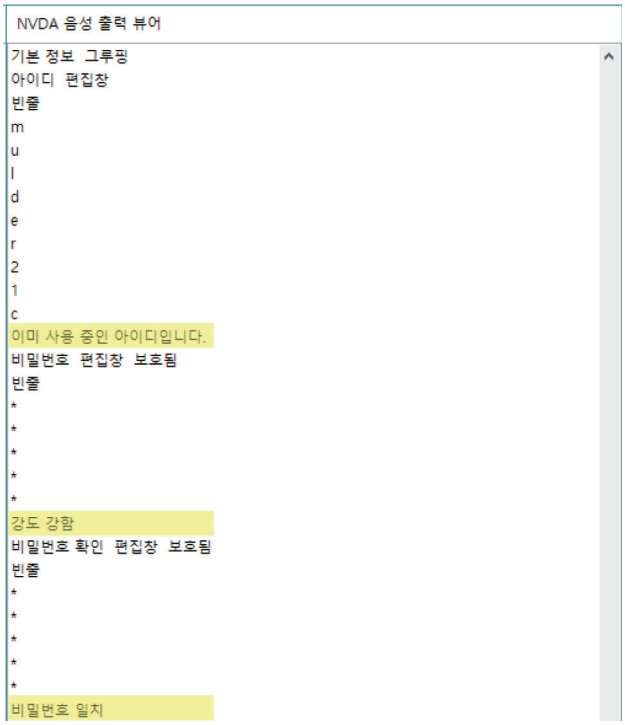


사용자에게 전달할 검증 결과 피드백을 포함하는 요소에 alert 역할(Role)을 추가해 주기만 하면 된다.

role="alert"으로 설정된 요소에 노드가 추가될 때마다 스크린리더는 해당 노드의 텍스트를 음성으로 출력하여 사용자로 하여금 현재 입력한 값에 대한 피드백을 실시간으로 전달 받을 수 있다.

참고로, alert 역할(Role)의 경우 이것은 이미 Live Region에 속하는 역할이기 때문에 aria-live 속성(Property)을 추가로 적용할 필요가 없다. 일부 최대 호환성을 위해 aria-live 속성(Property)을 적용하는 사용자들도 있기는 하지만, 이렇게 되었을 경우 iOS의 Voice Over에서 중복하여 읽어주는 이슈가 있는 것으로 알려져 있으니 참고하도록 하자.

WAI-ARIA 적용에 따른 스크린리더 음성 출력 차이

WAI-ARIA 적용 이전	WAI-ARIA 적용 이전
	

• WAI-ARIA 속성 정리

각 요소에 적용되는 역할(Role)을 정리해보면 다음과 같다.

요소	역할	속성/상태	설명
div	alert		사용자의 즉각적인 주의가 필요한 정보 전달

• 소스 코드 정리

이제, 앞서 설명한 대로 WAI-ARIA를 적용 시켜 보도록 하자.

앞서 언급한 대로 피드백이 노출될 요소에 role="alert"을 추가하면 된다. 예제에서는 입력 상자의 형제 요소로 중립적인 의미를 가지는 〈div〉 요소를 사용했다

```
<form action="#" name="frm2" id="frm2" method="post">
    <fieldset>
        <legend class="hidden-accessible">기본 정보</legend>
        <div class="field">
            <label for="usrId">아이디</label>
            <input type="text" name="usrId" id="usrId" >
            <div class="alert" role="alert" ></div>
        </div>
        <div class="field">
            <label for="usrPwd">비밀번호</label>
            <input type="password" name="usrPwd" id="usrPwd" >
            <div class="alert" role="alert" ></div>
        </div>
        <div class="field">
            <label for="reUsrPwd">비밀번호 확인</label>
            <input type="password" name="reUsrPwd" id="reUsrPwd" >
            <div class="alert" role="alert" ></div>
        </div>
    </fieldset>
</form>
```

이 role="alert"이 적용된 요소에 피드백을 추가하는 것은 단지 사용자에게 전달할 텍스트를 담고 있는 〈div〉 요소(Element)를 생성하여 추가하기만 하면 된다.

예를 들어 ID를 검증할 경우, 사용자의 입력 값에 대한 검증 후 검증 결과에 따라 피드백을 추가하는 코드는 다음과 같다.

```
// ID 검증
$('input[name^="usrId"]').on({
    'focus' : function(){
        orgVal = '';
    },
    'keyup' : function(){
        clearTimeout(tm);
        var me = this;
        // 입력된 값이 없거나, 키 입력이 끝났을 때
        // 기존 값과 현재 값이 동일하면 검증하지 않음
        if( me.value.trim() === '' || me.value.trim() === orgVal){
            return;
        }
        tm = setTimeout(function(){
            // ID 검증을 위한 ajax 호출
            $.ajax({
                url : '...',
                data : { id : me.value },
                method : 'POST',
                dataType : 'json',
                cache : false
             })
             .done(function(data){
                 // 피드백 추가
                 feedback($(me).find(' ~ .alert:eq(0)'), data['result']);
             })
        }, 400);
    }
});
```

여기서 핵심이 되는 것은 feedback 함수인데, 본 예제에서는 피드백을 추가할 대상 요소와 추가할 내용 유형을 인자로 전달 받아 해당 요소에 추가하도록 하였다.

```
// 피드백 메세지 정의
var feedbackMsg = {
    'duplicated' : '이미 사용 중인 아이디입니다.',
    'blocked' : '사용 불가한 아이디 입니다.',
    'passed' : '사용 가능한 아이디 입니다.',
    'shortid' : '아이디가 너무 짧습니다',
    'strength' : ['very-weak', 'weak', 'normal', 'strong', 'very-strong'],
    'very-weak' : '강도 매우 취약',
    'weak' : '강도 취약',
    'normal' : '강도 보통',
    'strong' : '강도 강함',
    'very-strong' : '강도 매우 강함',
    'same' : '비밀번호 일치',
    'non-same' : '비밀번호 일치하지 않음'
};
// 피드백 메세지 노출 함수
function feedback(elem, feedbackCode){
    $(elem)
     .empty().append('<div class="' + feedbackCode + '">' +
feedbackMsg[feedbackCode] + '</div>');
}
```

feedback 함수의 호출에 따라 해당 피드백 요소는 다음과 같이 변경된다.

```
<div class="field">
    <label for="usrId2">아이디</label>
    <input type="text" name="usrId2" id="usrId2">
    <div class="alert" role="alert">
        <div class="duplicated">이미 사용 중인 아이디입니다.</div>
    </div>
</div>
```

다음으로는 비밀번호의 강도를 측정하여 측정 결과에 따라 피드백을 추가하는 코드를 작성하도록 한다. 예제에서는 비밀번호의 강도를 측정에 zxcvbn.js 라이브러리를 사용하여 측정되도록 작성하였다.

```
$('input[name^="usrPwd"]').on({
    'focus' : function(){
        orgVal = '';
    },
    'keyup' : function(event){
        var me = this;
        // 입력된 값이 없거나, 키 입력이 끝났을 때
        // 기존 값과 현재 값이 동일하면 검증하지 않음
        if( me.value.trim() === '' || me.value.trim() === orgVal ){
            return;
        }
        clearTimeout(tm);
        tm = setTimeout(function(){
            orgVal = me.value.trim();
            // zxzxcvbn 결과에 따른 피드백 추가
            feedback(
                $(me).find(' ~ .alert:eq(0)'),
                feedbackMsg['strength'][zxcvbn(orgVal)['score']]
            );
        }, 400);
    }
});
```

마지막으로 비밀번호 확인에 대한 검증 부분이다. 비밀번호 확인은 비밀번호 입력 값과 동일 여부를 판단하여 피드백을 추가하면 된다.

```
// re-PWD 검증
$('input[name^="reUsrPwd"]').on({
    'keyup' : function(event){
        var me = this;
        // 입력된 값이 없거나, 키 입력이 끝났을 때
        // 기존 값과 현재 값이 동일하면 검증하지 않음
        if( me.value.trim() === '' || me.value.trim() === orgVal ){
            return;
        }
        clearTimeout(tm);
        tm = setTimeout(function(){
            // 비밀번호 - 비밀번호 확인 간 일치 여부에 따른 피드백 추가
            feedback(
```

```
                $(me).find(' ~ .alert:eq(0)'),
                me.value.trim() === $('#' + me.getAttribute('data-origin'))
                                        .val() ? 'same' : 'non-same'
            );
        }, 400);
    }
});
```

20. 캐러셀(Carousel) UI

• 기존 코드의 문제점들

캐러셀(Carousel, 화면 슬라이드) 유형의 UI는 웹 사이트/애플리케이션에서 자주 사용되는 컴포넌트로, 한정된 컴포넌트 영역 내에서 많은 콘텐츠를 슬라이드로 보여줄 수 있어 매우 효과적이다.

NIKE.com 캐러셀 UI

그러나 많은 웹 사이트/애플리케이션에서 제공하는 캐러셀 UI의 접근성을 살펴보면 전맹, 저시력 등의 스크린리더 사용자, 키보드에 의존하는 지체 장애자 분들을 위한 배려를 찾아보기란 쉽지 않다.

정보 접근에 혼란이 생기거나 정보에 대한 이해가 어려우니 서비스를 제대로 이용할 수 조차 없다. 장애를 이유로 이와 같은 정보에 대한 접근이 차별 되어서는 안 된다. 여기서 명심할 것은 UI 제작 과정에서 접근성을 반드시 고려하여야 한다는 것이다.

지금부터 일반적으로 사용되는 캐러셀 UI 구현 코드와 유사한 구조를 띈 국내 유명 브랜드 업체의 실제 서비스 구현 사례를 살펴보면서 몇몇 접근성 문제를 짚어보자.

국내 유명 브랜드 업체의 캐러셀 UI 마크업

```
<div class="main_slide">
    <ul id="mainSlide">
        <li>
           <a href="자바스크립트: openMovie('...');"
              style="background: url('...') no-repeat 50% 0px;">
                <div class="desktop"><img alt="" src="/MAIN_d.jpg"></div>
                <div class="mobile"><img alt="" src="/MAIN_m.jpg"></div>
           </a>
        </li>
        <li>
           <a href="..." target="_self"
              style="background:url('...') 50% 0 no-repeat">
                <div class="desktop">
                     <img src="/20160119AM94649_6884.jpg" alt="" />
                </div>
                <div class="mobile">
                     <img src="/20160119AM94649_9794.jpg" alt="" />
                </div>
           </a>
        </li>
        <li>
           <a href="..."
              style="background: url('...') no-repeat 50% 0px;">
                <div class="desktop">
                     <img alt="" src="20160119AM94725_7994.jpg">
                </div>
                <div class="mobile">
                     <img alt="" src="20160119AM94725_7789.jpg">
                </div>
           </a>
        </li>
    </ul>
</div>
```

문서의 구조를 살펴보면 목록과 하이퍼링크, 이미지 요소들로 구성되어 있다. 한 눈에 봐도 사용자에게 제공되는 텍스트 콘텐츠는 찾아보기 힘들다. 비 장애인과 달리 시각 장애를 가진 사용자는 정보 접근에 차별을 받게 된다.

그렇다면 자바스크립트 코드가 처리하여 동적으로 변경한 마크업 코드는 접근성에 문제가 없도록 반영되었을까?

자바스크립트 플러그인이 동적으로 변경한 캐러셀 UI 마크업

```
<div class="main_slide">
    <div class="bx-wrapper" style="max-width: 100%;">
       <div class="bx-viewport"
            style="width: 100%;
                   overflow: hidden;
                   position: relative;
                   height: 222px;">
          <ul id="mainSlide"
              style="width: 515%;
                     position: relative;
                     transition-duration: 0s;
                     transform: translate3d(-789px, 0px, 0px);">
             <li class="bx-clone"
                 style="float: left;
                        list-style: none;
                        position: relative;
                        width: 263px;">
                 <a href="..."
                    style="background: url(/20160119AM94725_7994.jpg)
                           no-repeat 50% 0px;">
                      <div class="desktop">
                           <img alt="" src="/20160119AM94725_7994.jpg">
                      </div>
                      <div class="mobile">
                           <img alt="" src="/20160119AM94725_7789.jpg">
                      </div>
                  </a>
              </li>
          ...
          </ul>
       </div>
```

```
        <div class="bx-controls bx-has-pager bx-has-controls-direction">
            <div class="bx-pager bx-default-pager">
                  <div class="bx-pager-item">
                        <a href="" data-slide-index="0"
                           class="bx-pager-link">1</a>
                  </div>
                  <div class="bx-pager-item">
                        <a href="" data-slide-index="1"
                           class="bx-pager-link">2</a>
                  </div>
                  <div class="bx-pager-item">
                        <a href="" data-slide-index="2"
                           class="bx-pager-link active">3</a>
                  </div>
            </div>
            <div class="bx-controls-direction">
                <a class="bx-prev" href="">Prev</a>
                <a class="bx-next" href="">Next</a>
            </div>
        </div>
    </div>
</div>
```

안타깝게도 자바스크립트 캐러셀 플러그인이 적용된 코드 역시 화면에 보여지는 콘텐츠에 대한 정보를 제공하지 않아 접근성에 문제가 있음을 보여준다. 특히 동적으로 추가 생성된 인디케이터(Indicator) 영역의 코드를 살펴보면 1, 2, 3, prev, next 정보만 제공할 뿐, 현재 활성화된 콘텐츠는 무엇이며 이전 또는 다음에 제공되는 콘텐츠에 대한 정보 등은 제공하고 있지 않다.

그렇다면 캐러셀 UI 컴포넌트를 제작하는데 있어 어떤 점을 사전에 고려하여야 할까?

스크린리더를 사용해야 하는 사람의 입장에서 생각해보면 캐러셀 UI에 접근했을 때 해당 컴포넌트 영역이 캐러셀 UI라는 정보를 사전에 제공받지 못했을 경우 혼란을 야기할 수 있으니 정확하게 인지할 수 있도록 정보를 구성해야 한다.

그리고 키보드 사용자 입장에서는 키보드만으로도 탐색 가능할 수 있어야 하며, 자동으로 재생되는 애니메이션 기능이 제공된다면 마우스가 아닌 키보드로 멈출 수 있는 기능을 제공해야 한다.

뿐만 아니라 UX(User eXperience) 관점에서 현재 활성화된 콘텐츠, 이전/다음 콘텐츠에 대한 정보를 이전/다음 버튼을 누르지 않고도 제공받을 수 있어야 한다.

• WAI-ARIA를 사용해야 하는 이유

WAI-ARIA를 적용하면 일반적인 HTML 구조 요소에 적절한 역할(Role)과 속성(Property), 상태(State)를 부여할 수 있어 스크린리더 사용자에게 의미적이고, 유효한 정보를 제공할 수 있을 뿐만 아니라, 업데이트 상태를 변경 순간마다 알려줄 수 있어 정확하게 현 상황에 대해 인지할 수 있도록 도와준다.

예를 들면 인디케이터에 사용된 〈a〉 요소를 '링크'가 아닌, '탭' 또는 '버튼'으로 읽어주도록 설정하거나, 탭과 탭 패널 역할이 부여된 요소 사이의 관계를 명시적으로 연결해주고, 화면에 보여지거나, 감춰진 상태를 변경될 때 마다 안내해주도록 설정할 수 있다.

• WAI-ARIA전체적인 그림과 설명

캐러셀 UI에 적용될 WAI-ARIA 역할(Role), 속성(Property), 상태(State) 구조를 살펴보면 다음과 같다.

역할은 role 속성으로, 속성 및 상태는 aria-* 유형으로 접두사가 붙는다. 상태 속성의 경우는 선택된 상태(aria-selected), 감춰진 상태(aria-hidden)가 사용된다.

캐러셀 UI에 적용된 WAI-ARIA 역할/속성/상태 구조

자! 그렇다면 이제부터 단계 별로 접근성을 준수한 캐러셀 UI 컴포넌트를 제작해보자.

먼저 캐러셀 UI 컴포넌트를 구성하는 HTML 구조부터 설계해본다. 캐러셀 UI는 슬라이딩되는 슬라이드를 감싸는 컨테이너임을 상기하여 구조화한다면 다음과 같이 디자인할 수 있다.

가장 일반적으로 사용되는 구조로 각 항목을 감싸는 목록을 생각할 수 있다. 그리고 이를 감싸는 역할을 수행하는 영역을 〈div〉 요소로 구성하는 것이다. 이때 컴포넌트를 재사용할 수 있도록 컴포넌트 식별자 역할을 담당할 class를 추가한다.

```
<div class="ui-carousel">
    <ul>
        <li class="ui-carousel__slide"></li>
        <li class="ui-carousel__slide"></li>
        <li class="ui-carousel__slide"></li>
    </ul>
</div>
```

다른 방법으로는 각각 독립 구성이 가능한 슬라이드 콘텐츠를 포함하는 영역을 생각할 수 있다. 여기서 각 슬라이드 영역은 〈article〉 요소로 마크업 하기로 하였다. 이때 해당 영역이 반드시 〈article〉 요소를 사용해야만 하는 것은 아니며 슬라이드 콘텐츠의 내용에 따라 적절한 요소로 대체 하여도 무관하다.

```
<div class="ui-carousel">
    <article class="ui-carousel__slide"></article>
    <article class="ui-carousel__slide"></article>
    <article class="ui-carousel__slide"></article>
</div>
```

화면에서 보이는 것만 생각하면 캐러셀 UI의 큰 틀을 만들어 낸 것처럼 생각할 수 있다. 하지만 눈에 보이는 것이 전부여서는 안 된다. 스크린리더 사용자의 입장에서 생각해보라. 저 구조로는 해당 영역이 캐러셀 UI 컴포넌트라는 정보를 알 수 없다는 사실을 말이다.

그렇다면 무엇이 필요할까? 눈에 보이지 않더라도 캐러셀 UI 컴포넌트 영역을 대표하는 제목이 필요하다는 사실을 떠올릴 수 있다. 캐러셀 영역에 제목을 추가하여 스크린리더 사용자에게 정확한 정보를 제공해야 한다. 단 화면에 보이지 않아야 하므로 스크린리더로는 읽을 수 있지만 화면에서 감춰지는 클래스 모듈 readable-hidden을 추가한다.

```
<div class="ui-carousel">
    <h2 class="ui-carousel__title readable-hidden">
        캐러셀을 기술하는 제목
    </h2>
    <article class="ui-carousel__slide">
        캐러셀 슬라이드 콘텐츠 내용 1
    </article>
    <article class="ui-carousel__slide">
        캐러셀 슬라이드 콘텐츠 내용 2
    </article>
    <article class="ui-carousel__slide">
        캐러셀 슬라이드 콘텐츠 내용 3
    </article>
</div>
```

Note.

스크린리더에서 읽을 수 있도록 콘텐츠를 감추는 방법에 대해 궁금하다면 "스크린리더 사용자를 위한 감춰진 콘텐츠"(goo.gl/6FxnRD)를 참고하라.

캐러셀 제목을 추가 함으로서 스크린리더 사용자에게 해당 컴포넌트에 대한 인식을 심어줄 수 있게 되었다. 하지만 이것만으로 충분한 것일까?

아무런 의미를 가지지 않는 〈div〉 요소로 그룹화 했다고 해서 캐러셀 UI 컴포넌트 영역(Region)으로서 구분이 되는 것은 아니다. WAI-ARIA는 이와 같은 문제를 개선하기 위해 "영역"을 의미하는 region 역할(Role) 속성을 제공한다. 작성한 구조에 region 역할을 부여해보자.

이어서 캐러셀 영역으로 구분된 〈div〉 요소에 aria-labelledby 속성(Property)을 추가한 후, 캐러셀의 제목에 해당하는 요소를 연결해준다. (id 속성 사용) aria-labelledby 속성은 요소에 레이블(Label)을 제공하기 위한 목적으로 사용된다. 이때 제공되는 레이블은 적절하고 간결해야 한다.

아래 작성된 코드를 살펴보자.

```
<div class="ui-carousel"
      role="region" aria-labelledby="my-carousel-title" >
    <h2 id="my-carousel-title" class="ui-carousel__title readable-hidden">
        캐러셀을 기술하는 제목
    </h2>
    <article class="ui-carousel__slide">
        캐러셀 슬라이드 콘텐츠 내용 1
    </article>
    <article class="ui-carousel__slide">
        캐러셀 슬라이드 콘텐츠 내용 2
    </article>
    <article class="ui-carousel__slide">
        캐러셀 슬라이드 콘텐츠 내용 3
    </article>
</div>
```

먼저 살펴본 방법은 기존 HTML 마크업에 스크린리더 사용자를 고려하여 제목 요소를 추가한 후, WAI-ARIA 역할과 속성을 추가한 것이다. 하지만 화면에 표시되지 않는 제목을 구성하여 추가한 후, CSS 스타일로 처리하는데 드는 수고가 적지 않은 것을 감안하면 그다지 효율적인 방법으로 생각되지 않을 수 있다.

예를 들어 기존에 작성된 캐러셀 코드를 찾아 제목 요소를 추가하고, 고유한 id 속성을 추가한 다음 레이블을 연결하기 위해 aria-labelledby 속성을 추가하는 것은 매우 번거롭다.

그렇다면 이보다 간편하면서 접근성을 향상하는 목적을 이루는 방법은 없을까?

WAI-ARIA의 속성 중 aria-label 속성을 사용하면 앞서 다룬 복잡하고 번거로운 과정을 손쉽게 해결할 수 있다. aria-label 속성은 화면에 표시되지는 않으면서 스크린리더에는 읽혀야 하는 콘텐츠를 제공할 때 매우 유용하게 사용될 수 있다.

아래 코드를 보라. 이전 코드보다 깔끔하고 간결하면서도 접근성을 향상하는 목적을 이룰 수 있다.

```
<div class="ui-carousel" role="region" aria-label="캐러셀을 기술하는 제목">
    <article class="ui-carousel__slide">
        캐러셀 슬라이드 콘텐츠 내용 1
    </article>
    <article class="ui-carousel__slide">
        캐러셀 슬라이드 콘텐츠 내용 2
    </article>
    <article class="ui-carousel__slide">
        캐러셀 슬라이드 콘텐츠 내용 3
    </article>
</div>
```

이어서 추가되는 HTML 구조화는 이후 자바스크립트 파트에서 동적으로 생성/추가될 것이지만, 구조를 작성하는 방법을 먼저 알아두어야 한다. 하드 코딩 과정을 통해 자바스크립트가 동적으로 생성/추가하는 코드를 살펴보자.

먼저 자바스크립트를 통해 동적으로 추가될 구조는 캐러셀 인디케이터(Carousel Indicator) 영역으로 캐러셀의 각 슬라이드 콘텐츠를 나타내거나, 이동할 수 있는 기능을 수행하는 인디케이터를 감싸는 컨테이너(Container)를 추가한다.

각 인디케이터는 슬라이드가 나열된 순서대로 화면에 표시되기에 순차 목록(Ordered List)을 사용한다. 그리고 해당 인디케이터는 목록의 개별 항목(List Item)마다 앵커(Anchor) 요소를 추가한 후, 화면에만 표시되는 지시자(아이콘)와 달리 스크린리더 사용자에게 손쉬운 이해를 도와줄 수 있도록 슬라이드 콘텐츠 설명을 감춤 클래스 모듈(readable-hidden)이 적용된 요소로 제공한다. 그리고 각각의 앵커 요소와 슬라이드 요소에 고유하게 식별 가능한 id 속성을 추가 함으로서 관계를 설정한다.

```
<div class="ui-carousel" role="region" aria-label="캐러셀을 기술하는 제목">
    <!-- 자바스크립트를 사용하여 동적으로 생성될 코드 -->
    <ol class="ui-carousel__indicators">
        <li>
            <a href="#ui-carousel__slide--01" class="ui-carousel__tab">
                <span class="readable-hidden">
                    슬라이드 1. 캐러셀 콘텐츠 제목
                </span>
            </a>
        </li>
        <li>
            <a href="#ui-carousel__slide--02" class="ui-carousel__tab">
```

```
                <span class="readable-hidden">
                    슬라이드 2. 캐러셀 콘텐츠 제목
                </span>
            </a>
        </li>
        <li>
            <a href="#ui-carousel__slide--03" class="ui-carousel__tab">
                <span class="readable-hidden">
                    슬라이드 3. 캐러셀 콘텐츠 제목
                </span>
            </a>
        </li>
    </ol>
    <!--// 자바스크립트를 사용하여 동적으로 생성될 코드 -->
    <article id="ui-carousel__slide--01" class="ui-carousel__slide">
        캐러셀 슬라이드 콘텐츠 내용 1
    </article>
    <article id="ui-carousel__slide--02" class="ui-carousel__slide">
        캐러셀 슬라이드 콘텐츠 내용 2
    </article>
    <article id="ui-carousel__slide--03" class="ui-carousel__slide">
        캐러셀 슬라이드 콘텐츠 내용 3
    </article>
</div>
```

우리는 앞서 순차 목록을 추가한 후 개별 항목에 앵커를 추가 함으로서 구조를 작성했다. 여기에 그럴듯 한 인디케이터 모양의 스타일을 추가하면 비 스크린리더 사용자는 인디케이터로 인식될 지 모르나, 스크린리더 사용자에게는 어디까지나 목록일 뿐이요, 앵커일 뿐이다. 요컨대 스크린리더 사용자에게도 비 장애인이 인식하는 것처럼 구조에 의미를 부여하여 정보 제공에있어 차별받지 않도록 설계하여야 한다.

WAI-ARIA의 역할(Role)을 사용하여 스크린리더 사용자도 쉽게 인식할 수 있도록 의미 구조화 해보자.

먼저 〈ol〉 요소에 탭 리스트(tablist) 역할인 role 속성을 사용하여 추가한다. 더 이상 순차 목록이 아닌, 탭 리스트 역할을 수행할 수 있게 구조화 되었다. 〈li〉 요소는 어디까지나 목록 항목을 구조화하는데 사용하는 것으로 탭 리스트 구조에서는 의미가 없으므로 role 속성을 presentation으로 설정한다. 그리고 〈a〉 요소에 role 속성을 tab으로 설정하여 탭의 역할을 수행하도록 설정한다.

```
<ol role="tablist" class="ui-carousel__indicators">
    <li role="presentation" >
        <a role="tab" href="#ui-carousel__slide--01"
           class="ui-carousel__tab">
            <span class="readable-hidden">
                슬라이드 1. 캐러셀 콘텐츠 제목
            </span>
        </a>
    </li>
</ol>
```

마지막으로 캐러셀 슬라이드 요소에 role 속성 값으로 tabpanel을 설정하여 각각의 tab에 연결되는 탭 패널 역할을 부여한다.

```
<article role="tabpanel" id="ui-carousel__slide--01" class="ui-carousel__slide">
    캐러셀 슬라이드 콘텐츠 내용 1
</article>
```

각각의 역할은 부여했으니 이어서 각각의 탭, 탭 패널 요소에 속성(Property), 상태(State)를 설정하여 관계를 형성하고, 사용자의 행동에 따라 변경되는 상황을 알려줄 수 있도록 설정한다.

먼저 탭 역할을 수행하는 〈a〉 요소에 탭 패널의 레이블로 연결할 수 있도록 id 속성을 추가한다. 이어서 탭이 조작하는 탭 패널을 명시적으로 설정하기 위해 aria-controls 속성을 추가한 후, 해당 탭 패널의 id 속성 값을 설정한다. 그리고 탭이 선택된 상태를 알릴 수 있도록 aria-selected 속성을 추가한다.

현재 상태는 선택이 된 상태이니 true 값을, 다른 탭이 선택이 되면 false 값으로 변경하도록 만들어 스크린리더 사용자에게 선택된 탭과 선택되지 않은 탭을 안내해줘야 한다. (상태 변경은 이벤트에 따라 자바스크립트로 처리한다)

또한 선택된 탭 이외에는 키보드 포커스가 이동하지 않도록 tabindex 속성을 설정한 후 값을 0으로 설정하고, 다른 탭에는 -1을 설정한다.

이어서 탭 패널 요소에는 자신을 조작하는 탭이 레이블 역할을 수행하도록 aria-labelledby 속성을 사용하여 연결된 탭 id 값을 설정한다. 그리고 화면에 표시되는 슬라이드 외에는 화면에서 감춰짐을 나타내는 상태 속성 aria-hidden 값을 설정하여 현재 보여지는 슬라이드는 false, 감춰진 슬라이드는 true 값을 설정한다.

```
<a
  role="tab"
  aria-controls="ui-carousel__slide--03"
  aria-selected="true"
  tabindex="0"
  id="ui-carousel__tab--03"
  href="#ui-carousel__slide--03"
  class="ui-carousel__tab">
  <span class="readable-hidden">
    슬라이드 3. 캐로셀 콘텐츠 제목
  </span>
</a>
<article
  role="tabpanel"
  aria-labelledby="ui-carousel__tab--03"
  aria-hidden="false"
  id="ui-carousel__slide--03"
  class="ui-carousel__slide">
  캐로셀 슬라이드 콘텐츠 내용 3
</article>
```

• WAI-ARIA 속성 정리

캐로셀 UI에 적용된 WAI-ARIA 역할(Role), 속성(Property), 상태(State) 등을 이해하기 쉽게 표로 정리해보자.

요소	역할	속성/상태	설명
〈div〉	region	aria-label	영역 역할을 설정하고 레이블 추가.
〈ol〉	tablist		tab의 집합 역할 설정.
〈li〉	presentation		〈li〉 요소가 가지는 리스트의 의미를 무시
〈a〉	tab	aria-selected="true | false"	tab 역할 설정 및 선택된 상태 처리.
		aria-controls="{}"	조작하려는 tabpanel의 ID 속성 값 설정.
		id="{}"	조작하려는 tabpanel이 참조하는 tab ID 속성 값.
〈article〉	role="tabpanel"	aria-hidden="true | false"	tabpanel 역할 설정 및 보조기기에서 들림/안들림 상태 처리
		aria-labelledby="{ }"	연결된 tab의 ID 속성 값 설정.
		id="{ }"	tab이 참조하는 tabpanel ID 속성 값.

혹시나 WAI-ARIA 속성을 다수 설정해야 한다는 점에서 난해함을 느끼고 있다면 큰 걱정은 하지 않아도 된다. 지금까지 우리가 작성한 WAI-ARIA 역할, 속성, 상태는 직접 하드 코딩 하지 않고, 자바스크립트가 동적으로 설정하도록 프로그래밍할 것이다.

사용자가 입력 작성하는 HTML 구조화는 최대한 단순하게 작성하는 것이 좋다. 복잡한 설계는 자바스크립트가 우리를 도와줄 것이다. 그 동안 HTML/CSS만 주로 다뤄왔다면 이 기회를 통해 자바스크립트를 공부해보길 권장한다.

• 소스 코드 정리

자바스크립트를 이용하여 앞서 다뤄봤던 WAI-ARIA 역할, 속성, 상태를 동적으로 추가 설정하도록 프로그래밍 해보자. 먼저 작성자가 하드 코딩 할 HTML 구조는 최대한 깔끔하고 단순하게 구성한다.

먼저 캐러셀 UI 컴포넌트로 설정할 〈div〉 요소에 고유 식별자 id를 설정한 다음, 캐러셀 UI 컴포넌트 레이블을 HTML5 data-* 접두사를 사용하여 식별 가능하도록 설정한다. 여기에 하드 코딩 된 레이블은 자바스크립트에서 aria-label 속성을 설정할 때 참고하게 된다.

이때 data-* 접두사 속성을 사용하지 않고 바로 aria-label 속성을 하드 코딩하는 것도 가능하다. 이어서 각 슬라이드 요소를 설명하는 레이블을 data-* 접두사 속성을 사용하여 작성한다. 여기에 작성한 레이블은 인디케이터의 텍스트로 설정되어 스크린리더 사용자에게 정보를 제공할 것이다.

```
<div id="yamoo9-carousel" data-label="캐러셀을 기술하는 제목">
    <article data-label="캐러셀 슬라이드 레이블 1">
        <a href="#">
            <img
              src="http:// placehold.it/400x240/16a085/fff/?text=Slide 01"
              alt="캐러셀 슬라이드 콘텐츠 1 내용">
        </a>
    </article>
    <article data-label="캐러셀 슬라이드 레이블 2">
        <a href="#">
            <img
              src="http:// placehold.it/400x240/16a085/fff/?text=Slide 02"
              alt="캐러셀 슬라이드 콘텐츠 2 내용">
        </a>
    </article>
    <article data-label="캐러셀 슬라이드 레이블 3">
        <a href="#">
```

```
            <img
              src="http:// placehold.it/400x240/16a085/fff/?text=Slide 03"
              alt="캐러셀 슬라이드 콘텐츠 3 내용">
        </a>
    </article>
</div>
```

작성될 자바스크립트는 jQuery 라이브러리 형태로 플러그인화 할 것이나, 설명을 위해 본문에서는 절차적으로 진행한다.

우리가 먼저 수행할 것은 캐러셀 UI가 적용될 요소를 문서에서 찾는 것, 인디케이터 코드를 동적으로 작성하는 것이다. 캐러셀 UI 컴포넌트가 될 요소의 id 또는 class 속성 식별자를 통해 문서에서 대상을 찾는다. 이어서 인디케이터 요소가 될 요소를 생성하고, 요소 내부에 코드로 삽입될 템플릿 코드를 작성한다.

```
(function(global, $) {
    'use strict';
    // 캐러셀 UI를 설정할 대상 요소를 jQuery 인스턴스 객체로 변수에 참조
    var $widget = $('#yamoo9-carousel');
    // 캐러셀 UI 탭 패널 요소 수집 후, jQuery 인스턴스 객체로 변수에 참조
    var $tabpanel = $widget.children();
    // 캐러셀 UI 인디케이터 컨테이너 요소 생성 후, 변수에 참조
    var $tablist = $('<ol role="tablist">');
    // 동적으로 생성될 인디케이터 템플릿
    var template_indicators = [
        '<li role="presentation">',
        '<a href="#" role="tab">',
        '<span class="readable-hidden"></span>',
        '</a>',
        '</li>'
    ].join('');
```

이어서 수집된 탭 패널의 개수만큼 반복 수행하여 인디케이터 컨테이너 내부에 아이템을 추가하는 프로그래밍을 작성해보자.

```
// 캐러셀 UI 탭 패널 요소를 순환한 후, 템플릿을 활용하여 동적으로 코드 생성
$.each($tabpanel, function(idx) {
    // 개별 탭 패널 참조
    var $panel = $tabpanel.eq(idx);
    // 탬플릿 문자열로 탭 요소 생성 후, 변수에 참조
    var $tab = $(template_indicators);
    // 탭 패널의 레이블 설정 값을 label 변수에 참조
    var label = $panel.attr('data-label');
    // 탭 내부에서 span요소를 찾아 label 속성 값을 콘텐츠로 설정
    // label 변수 값이 존재하지 않으면,
     // 패널 내부의 제목(첫번째 매칭 요소) 콘텐츠를 설정
    // 제목 또한 존재하지 않으면 '슬라이드 N'으로 설정
    $tab.find('span').text(label||$panel.find(':header:eq(0)').text()||'슬라이드
'+(idx+1));
    $tab.attr('title', label||$panel.find(':header:eq(0)').text()||'슬라이드
'+(idx+1));
    // 완성된 코드는 $tablist에 마지막 자식 요소로 삽입
    $tab.appendTo($tablist);
});
// 완성된 $tablist 요소를 $widget 요소의 첫번째 자식 요소로 삽입힌다.
$widget.prepend($tablist);
```

지금까지 작성된 코드의 결과를 웹 브라우저 개발 도구(DevTool)를 통해 살펴보면 다음과 같다. 캐러셀 UI 인디케이터 코드가 슬라이드 콘테츠 보다 앞서 추가된 점을 눈 여겨 보자.

자바스크립트에 의해 동적으로 생성된 캐러셀 인디케이터

```
▼<div id="yamoo9-carousel" data-label="캐러셀을 기술하는 제목">
  ▼<ol role="tablist">
    ▼<li role="presentation">
      ▼<a href role="tab">
          <span class="readable-hidden">캐로셀 슬라이드 레이블 1</span>
        </a>
      </li>
    ▶<li role="presentation">…</li>
    ▶<li role="presentation">…</li>
    </ol>
  ▶<article data-label="캐러셀 슬라이드 레이블 1">…</article>
  ▶<article data-label="캐러셀 슬라이드 레이블 2">…</article>
  ▶<article data-label="캐러셀 슬라이드 레이블 3">…</article>
  </div>
```

다음은 이전/다음 슬라이드 보기 버튼을 추가하는 코드를 작성한다.

각 버튼은 슬라이드의 좌우에 위치한 버튼으로, 생성 할 때 식별되는 class 속성을 추가하도록 한다. 그리고 내부에는 화면에 보이지는 않지만, 스크린리더 사용자에게 이전/다음 슬라이드가 어떤 내용인지 정보를 제공해 줄 수 있는 감춤 콘텐츠 요소를 추가한다.

현재는 제공할 정보가 비워져 있는데, 이는 상태 변경에 따라 자동으로 이전/다음 레이블 값을 받아와서 처리되는 영역이 된다. 뒤에서 이 부분을 프로그래밍 해볼 것이다.

```
// 이전/다음 슬라이드 보기 버튼 추가
$.each(['prev', 'next'], function(idx, feature) {
    $('<button type="button" class="ui-carousel__button ui-carousel__but-
ton--'+feature+'">')
        .html('<span class="readable-hidden"></span>').appendTo($widget);
});
```

작성된 코드 결과를 웹 브라우저 개발 도구에서 확인해보면 다음과 같이 동적으로 코드가 생성된다.

자바스크립트에 의해 동적으로 생성된 이전/다음 버튼

```
▼<div id="yamoo9-carousel" data-label="캐러셀을 기술하는 제목">
  ▶<ol role="tablist">…</ol>
  ▶<article data-label="캐러셀 슬라이드 레이블 1">…</article>
  ▶<article data-label="캐러셀 슬라이드 레이블 2">…</article>
  ▶<article data-label="캐러셀 슬라이드 레이블 3">…</article>
  ▼<button type="button" class="ui-carousel__button ui-carousel__button--prev">
     <span class="readable-hidden"></span>
   </button>
  ▶<button type="button" class="ui-carousel__button ui-carousel__button--next">…</button>
 </div>
```

이로서 동적으로 생성 추가되어야 할 HTML 구조는 마무리되었다.

이어서 다뤄볼 내용은 WAI-ARIA 역할, 속성, 상태 및 식별 가능한 id, class 속성을 추가해볼 것이다.

먼저 $widget 요소에 class 속성 식별자 'ui-carousel', 역할 'region', 속성 'aria-label'을 각각 추가한다. (앞에서 다룬 예에서 그랬듯이 작성자가 data-* 접두사 속성을 사용하지 않았을 경우를 대비한 대안을 코드에 추가했다) 그리고 $tablist 요소에 class 속성 식별자 'ui-carousel__indicators'를 추가한다.

```
// $widget 요소에 클래스 속성 식별자 및 WAI-ARIA 적용
$widget.attr({
    // 클래스 식별자
    'class': 'ui-carousel',
    // 역할
    'role': 'region',
    // 속성
    'aria-label': $widget.attr('data-label') || $widget.children(':head-
er:eq(0)').text() || '캐로셀 UI: 슬라이드 메뉴'
});
// 탭 리스트 요소에 클래스 속성 식별자 추가
$tablist.addClass('ui-carousel__indicators');
```

다음 단계로 $tablist 인디케이터 영역에서 role 속성 값이 tab인 요소를 수집한 후 반복 순환하며 id, class 식별자와 WAI-ARIA 속성을 설정한다. 이때 각 속성은 HTML 구조 파트에서 동적으로 생성될 코드라며 소개했던 내용을 그대로 답습하여 추가한다.

각 슬라이드를 구분하는 식별자는 숫자로 구분하되, 10보다 작은 숫자의 경우 0을 앞에 붙여서 추가하도록 설정한다. 10보다 작은 수에 0을 붙여 반환하는 기능을 수행하는 헬퍼 함수 readingZeroNum() 코드는 뒤이어 작성해본다. 그리고 각 탭의 기본 선택 유무 상태는 false로, 각 슬라이드의 화면 감춤 상태는 true로 기본 설정한다.

```
// 탭 리스트 내부 탭에 클래스 속성 식별자 추가
var $tabs = $tablist.find('[role="tab"]');
$.each($tabs, function(idx) {
    // 개별 탭 참조
    var $tab = $tabs.eq(idx);
    // 0을 붙인 숫자로 변경하는 함수를 사용하여 반환된 결과를 num 변수에 참조
    var num = readingZeroNum(idx);
    var slide_id = 'ui-carousel__slide--' + num;
    $tab.attr({
        'id': 'ui-carousel__tab--' + num,
        'class': 'ui-carousel__tab',
        'aria-controls': slide_id,
        'aria-selected': false,
        'tabindex': -1
    });
    // 탭 패널에 클래스 속성 식별자 추가
```

```
    var $panel = $tabpanel.eq(idx);
    $panel.attr({
        'class': 'ui-carousel__tabpanel',
        'id': slide_id,
        'role': 'tabpanel',
        'aria-labelledby': 'ui-carousel__tab--01',
        'aria-hidden': true
    });
});
```

10보다 작은 수에 0을 붙여 반환하는 헬퍼 함수 코드는 다음과 같다.

```
/**
 * @function readingZeroNum
 * @param  {number} idx
 * @return {string | number}
 */
function readingZeroNum(idx) {
    var num = idx + 1;
    return 10 > num ? '0' + num : num;
}
```

이어서 웹 브라우저 개발 도구를 통해 생성된 코드를 살펴 보자. 우리가 원하는 대로 동적으로 코드가 잘 설정된 것을 확인할 수 있다. 앞에서도 이야기 했었지만, 자바스크립트를 사용하면 이와 같이 복잡한 WAI-ARIA 구조 코드를 자동으로 설정되도록 만들 수 있다.

자바스크립트에 의해 동적으로 생성된 속성

```
▼<div id="yamoo9-carousel" data-label="캐러셀을 기술하는 제목" class="ui-carousel" role="region"
aria-label="캐러셀을 기술하는 제목">
  ▼<ol role="tablist" class="ui-carousel__indicators">
    ▼<li role="presentation">
      ▶<a href="#" role="tab" id="ui-carousel__tab--01" class="ui-carousel__tab" aria-controls=
      "ui-carousel__slide--01" aria-selected="false" tabindex="-1">…</a>
      </li>
    ▶<li role="presentation">…</li>
    ▶<li role="presentation">…</li>
    </ol>
  ▶<article data-label="캐러셀 슬라이드 레이블 1" class="ui-carousel__tabpanel" id="ui-
  carousel__slide--01" role="tabpanel" aria-labelledby="ui-carousel__tab--01" aria-hidden="true">
  …</article>
  ▶<article data-label="캐러셀 슬라이드 레이블 2" class="ui-carousel__tabpanel" id="ui-
  carousel__slide--02" role="tabpanel" aria-labelledby="ui-carousel__tab--01" aria-hidden="true">
  …</article>
  ▶<article data-label="캐러셀 슬라이드 레이블 3" class="ui-carousel__tabpanel" id="ui-
  carousel__slide--03" role="tabpanel" aria-labelledby="ui-carousel__tab--01" aria-hidden="true">
  …</article>
  ▶<button type="button" class="ui-carousel__button ui-carousel__button--prev">…</button>
  ▶<button type="button" class="ui-carousel__button ui-carousel__button--next">…</button>
  </div>
```

동적으로 생성되는 코드의 마무리 파트로 슬라이드를 감싸는 요소를 동적으로 추가하는 코드를 작성한다.

```
// 슬라이드를 감싸는 영역을 동적으로 생성
$tabpanel.wrapAll('<div class="ui-carousel__tabpanel-wrapper">');
```

탭 패널 영역을 감싸는 래퍼 요소

```
// 슬라이드를 감싸는 영역을 동적으로 생성
$tabpanel.wrapAll('<div class="ui-carousel--tabpanel-wrapper">');

▼<div id="yamoo9-carousel" data-label="캐러셀을 기술하는 제목" class="ui-carousel" role="region"
aria-label="캐러셀을 기술하는 제목">
  ▶<ol role="tablist" class="ui-carousel__indicators">…</ol>
  ▼<div class="ui-carousel--tabpanel-wrapper">
    ▶<article data-label="캐러셀 슬라이드 레이블 1" class="ui-carousel__tabpanel" id="ui-
    carousel__slide--01" role="tabpanel" aria-labelledby="ui-carousel__tab--01" aria-hidden="true">
    …</article>
    ▶<article data-label="캐러셀 슬라이드 레이블 2" class="ui-carousel__tabpanel" id="ui-
    carousel__slide--02" role="tabpanel" aria-labelledby="ui-carousel__tab--01" aria-hidden="true">
    …</article>
    ▶<article data-label="캐러셀 슬라이드 레이블 3" class="ui-carousel__tabpanel" id="ui-
    carousel__slide--03" role="tabpanel" aria-labelledby="ui-carousel__tab--01" aria-hidden="true">
    …</article>
    </div>
  ▶<button type="button" class="ui-carousel__button ui-carousel__button--prev">…</button>
  ▶<button type="button" class="ui-carousel__button ui-carousel__button--next">…</button>
  </div>
```

캐러셀 UI 컴포넌트를 구성하는 동적 구조가 완성되었으니, 이를 사용자에게 캐러셀 UI로 보이도록 CSS 스타일링 한다. 여기서는 보편적으로 사용되는 캐러셀 UI와 같은 패턴으로 CSS 스타일링 해보았다.

CSS 디자인이 적용된 캐러셀 View

우리는 앞서 초기화 과정을 통해 뷰(View)를 구현해보았다. 이제는 각 컨트롤 요소에 기능을 추가해보도록 하자. 인디케이터 영역의 탭 버튼에 이벤트/이벤트 핸들러를 연결해볼 것이다.

```
// $tabs를 반복 순환하여 개별 $tab 요소에
// 이벤트 ↔ 이벤트 핸들러 바인딩
$.each($tabs, function(idx) {
    // 개별 $tab 참조
    var $tab = $tabs.eq(idx);
    // $tab 요소에 이벤트 핸들러 바인딩
    // $.proxy() 프록시 우회 메소드를 사용하여
    // this 컨텍스트 참조 변수 설정
    $tab.on('click', $.proxy(activeSlide, $tab));
});
```

이어서 $tab 요소 이벤트에 연결될 함수 activeSlide()를 정의 한다. 각 탭은 〈a〉 요소에 role="tab"을 설정하여 탭의 역할을 부여했으나, 클릭할 경우 브라우저의 기본 동작이 수행되므로 이를 먼저 차단할 필요가 있다.

그리고 사용자가 탭을 클릭할 때마다 업데이트가 되어야 하는 사항(선택, 감춤)을 각각 분리하여 함수 호출하는 코드를 추가한다. 각 함수의 이름은 명시적이고 직관적인 이름을 사용하여 작성한다.

```
/**
 * @function activeSlide
 * @param  {event} e
*/
function activeSlide(e) {
    // 브라우저 기본 동작 차단
    e.preventDefault();
    var index = getIndex.call(this);
    // 사용자 상호작용에 따른 "선택 상태" 변경 메소드 호출
    changeStateSelect.call(this);
    // 사용자 상호작용에 따른 "감춤 상태" 변경 메소드 호출
    changeStateHidden.call(this);
    // 1. 탭 패널의 부모 요소(<ol>)의 left 위치 값을 선택된 탭의 인덱스에 따라 변경
    // $tabpanel.parent().css('left', $tabpanel.outerWidth() * index * -1);
    // 2. 위 코드를 애니메이션 기능 적용 예로 바꾼 코드
    $tabpanel.parent().stop().animate({
        'left': $tabpanel.outerWidth() * index * -1
    });
    // 버튼 텍스트 업데이트
    updateButtonText(index);
}
```

activeSlide() 함수가 호출하는 changeStateSelect() 함수를 정의해보자. 이 함수가 참조하는 this 키워드는 사용자가 선택한 탭이다. 탭을 사용자가 누르면 눌러진 탭은 활성화가 되고, 다른 탭은 비활성화가 되도록 상태를 업데이트 하는 코드를 작성한다. aria-selected, tabindex 값이 변경되어야 하며 active 클래스 속성 설정 값도 업데이트가 되어야 한다.

```
/**
 * @function changeStateSelect
*/
function changeStateSelect() {
    // 선택된 탭의 부모 형제의 자식 중 role 속성 값이 tab 인 요소를 찾아
    // WAI-ARIA 상태 변경
    // 키보드 포커싱이 되지 않도록 설정
    // active 클래스 속성 제거
    this.parent().siblings().find('[role="tab"]').attr({
        'aria-selected': false,
        'tabindex': -1
    }).removeClass('active');
```

```
    // 선택된 탭 요소의 WAI-ARIA 상태 변경
    // 키보드 포커싱이 되도록 설정
    // active 클래스 속성 추가
    this.attr({
        'aria-selected': true,
        'tabindex': 0
    }).addClass('active');
}
```

이어서 changeStateHidden() 업데이트 함수도 작성해본다.

이 함수가 전달받은 this 컨텍스트가 참조하는 대상은 사용자가 선택한 탭이다. 선택한 탭의 aria-controls 속성 정보 값을 통해 제어할 탭 패널(슬라이드)을 찾은 후, 형제 탭 패널은 감춤 상태를 감춤으로 활성화 된 탭 패널은 감춤 상태를 보임 상태로 변경한다.

```
/**
 * @function changeStateHidden
*/
function changeStateHidden() {
    // 사용자가 선택한 탭의 aria-controls 속성 값을 통해
    // 제어해야 할 탭 패널(슬라이드)를 필터링한다.
    var $panel = $tabpanel.filter('#' + this.attr('aria-controls'));
    // 필터링된 탭 패널의 형제 요소 중, aria-selected 속성을 가진 요소를 찾아
    // 감춤 상태를 업데이트 한다.
    $panel.siblings(['aria-selected']).attr({
        'aria-hidden': true
    }).find('a').attr('tabindex', -1);
    // 사용자가 선택한 탭이 제어하는 탭 패널의 감춤 상태를 업데이트 한다.
    $panel.attr({
        'aria-hidden': false
    }).find('a').removeAttr('tabindex');
}
```

다음으로 activeSlide() 함수에서 사용된 getIndex() 함수를 정의한다. getIndex() 함수 내부의 this 컨텍스트 참조는 사용자가 선택한 탭이다. 탭의 aria-controls 속성 값에서 인덱스 정보 값을 가져와 숫자 유형으로 형 변환 시킨 후, 값을 반환하도록 구성한다.

```
/**
 * @function getIndex
 * @return  {number}
*/
function getIndex() {
    // 선택된 탭의 aria-controls 속성 값에서 인덱스 정보를 뽑아 반환한다.
    return Number(this.attr('aria-controls').split('--')[1]) - 1;
}
```

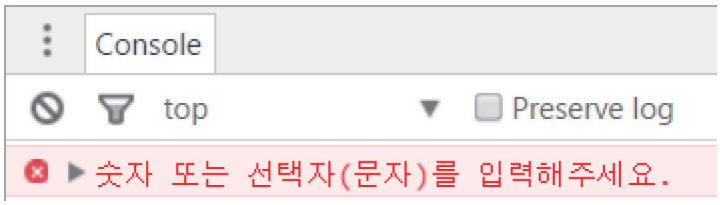
이어서 빈번하게 재사용 될 함수 activeTab()을 정의한다. 앞서 다룬 함수들과 달리 이 함수는 인자를 전달 받아 처리한다. 인자 값으로는 숫자 또는 문자 유형에 따라 활성화 할 탭을 필터링 하여 해당 탭을 활성화하도록 작성한다. 만약 인자 값이 주어진 조건인 숫자 또는 문자 유형이 아니라면 오류 메시지를 콘솔에 출력하도록 설정하고 함수를 종료하게 한다.

```
/**
 * @function activeTab
 * @param  {string | number} id
*/
function activeTab(id) {
  // 전달된 id 인자 값에 따라 활성화할 탭 필터링
  var $filter, type = $.type(id);
  // 숫자일 경우, 해당 숫자에 해당하는 탭 활성화
  if (type === 'number') {
      $filter = $tabs.eq(id - 1);
  }
  // 문자일 경우, 해당 선택자에 해당하는 탭 활성화
  else if (type === 'string') {
      $filter = $tabs.filter(id);
  }
  // 숫자,문자가 아닐 경우 오류 메시지 출력 후, 함수 종료
  else {
      return console.error('숫자 또는 선택자(문자)를 입력해주세요.');
  }
  // 필터링 된 탭 활성화(클릭)
  $filter.trigger('click');
}
```

정의된 activeTab() 함수를 사용하는 방법은 활성화 할 탭의 인덱스 순서를 전달하거나, 탭의 선택자를 문자열로 전달하면 된다. 만약 다른 유형을 전달할 경우 오류를 발생시킨 후 함수를 종료하게 된다.

```
// 활성화 할 탭 인덱스를 설정
activeTab(2);
```

오류가 발생한 경우, 콘솔 패널에 오류 메시지 표시



탭 활성화와 관련된 이벤트 핸들링 기능은 마무리 되었으니 다음 단계로 이전/다음 버튼 기능을 프로그래밍 해보자.

```
// 이전/다음 슬라이드 탐색 버튼을 찾아 변수 $buttons에 참조
var $buttons = $('.ui-carousel__button');
// $buttons 요소에 click 이벤트 핸들러 바인딩
// 이벤트 핸들러 activeTabWithButton
$buttons.on('click', activeTabWithButton);
```

이어서 activeTabWithButton() 함수를 정의해본다. 활성화된 탭을 통해 인덱스를 알아내어 제어하도록 제작한다.

```
/**
 * @function activeTabWithButton
*/
function activeTabWithButton() {
    // 활성화 된 탭을 찾아 $tab 변수에 참조
    var $tab = $tabs.filter('.active');
    // getIndex 함수를 사용하여 활성화 된 탭의 인덱스 값 index 변수에 참조
    var index = getIndex.call($tab) + 1;
```

```
    // 이전 버튼일 경우 true 값이 isClickPrevBtn 변수에 참조
    var isClickPrevBtn = this.getAttribute('class').indexOf('prev') > -1;
    // $tabs의 개수 값을 length 변수에 참조
    var length = $tabs.length;
    // isClickPrevBtn 참조 값이 참일 경우 실행
    if (isClickPrevBtn) {
        // 유효한 index 값을 설정함
        index = --index > 0 ? index : length;
    }
    // isClickPrevBtn 참조 값이 거짓일 경우 실행
    else {
        // 유효한 index 값을 설정함
        index = ++index <= length ? index : 1;
    }
    // 해당 index에 해당하는 탭 활성화
    activeTab(index);
}
```

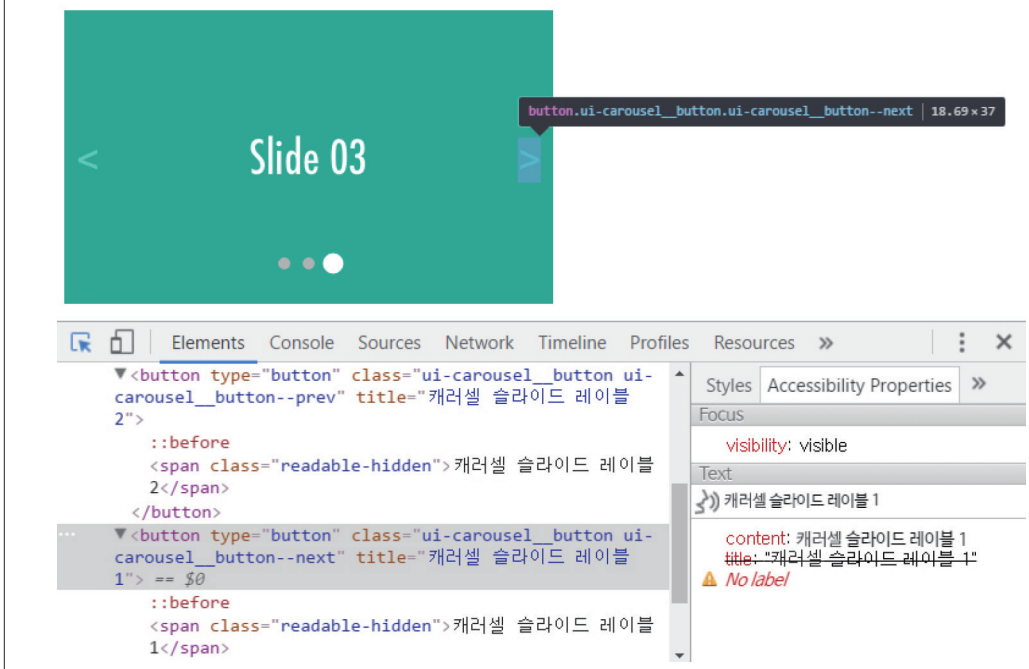
앞서 activeSlide() 함수에서 사용된 updateButtonText() 함수를 정의해보자. 이 함수는 사용자가 버튼을 클릭할 때마다 이전/다음 버튼의 콘텐츠를 자동으로 바꿔주는 역할을 수행한다.

```
/**
 * @function updateButtonText
 * @param  {number} idx
*/
function updateButtonText(idx) {
    // 전달받은 idx 값을 사용하여 활성화된 탭을 $tab 변수에 참조
    var $tab = $tabs.eq(idx - 1);
    // getIndex() 함수를 사용하여 활성화된 탭의 index 값을 참조
    var index = getIndex.call($tab);
    // $buttons 객체에 수집된 요소 중 이전 버튼을 필터링하여 참조
    var $prevBtn = $buttons.filter('.ui-carousel__button--prev');
    // $buttons 객체에 수집된 요소 중 다음 버튼을 필터링하여 참조
    var $nextBtn = $buttons.filter('.ui-carousel__button--next');
    // 이전 버튼에 삽입되어야 할 텍스트를 탭 내부의 span 요소에서 참조
    var prevText = $tabs.eq(index - 1).find('span').text();
    // 다음 버튼에 삽입되어야 할 텍스트를 탭 내부의 span 요소에서 참조
    var nextText = $tabs.eq(index + 1 === 3 ? 0 : index + 1).find('span').
text();
```

```
    // 이전 버튼 내부의 span 요소에 prevText 값 설정
    $prevBtn.find('span').text(prevText);
    // 이전 버튼의 title 속성에 prevText 값 설정
    $prevBtn.attr('title', prevText);
    // 다음 버튼 내부의 span 요소에 nextText 값 설정
    $nextBtn.find('span').text(nextText);
    // 다음 버튼의 title 속성에 nextText 값 설정
    $nextBtn.attr('title', nextText);
}
```

코드를 마무리 한 후, 웹 브라우저 개발 도구를 통해 동적으로 변경된 버튼 텍스트 코드를 살펴볼 수 있다. 현재 활성화된 슬라이드의 인덱스는 세 번째이고, 이전 슬라이드는 두 번째, 다음 슬라이드는 첫 번째 슬라이드 임을 정확하게 처리하는 결과를 확인할 수 있다.

updateButtonText() 함수에 의해 자동으로 설정된 버튼 텍스트



이어서 키보드 방향키로 탭 간 탐색을 가능하도록 설정하는 기능을 프로그래밍해보도록 하자. 사용자가 누른 키를 감지하는 조건 문을 작성한 후, 각 조건에 맞는 유효한 인덱스 값을 설정하도록 코드를 작성한다.

만약 사용자가 다른 키를 누르면 동작하지 않도록 조치하여 방향 키만으로 조작할 수 있도록 제한한다. activeTab() 함수에 유효한 index 값을 전달해 방향키로 탭 간 탐색 및 활성화를 가능하도록 설정한다. 그리고 포커스도 이동될 수 있도록 활성화된 탭에 포커스를 적용한다.

```
// 키보드로 제어하는 탭 내비게이션 활성화를 위한 이벤트 핸들링
$tabs.on('keydown', activeKeyboardNavigation);
// @function activeKeyboardNavigation, @param  {event} e
function activeKeyboardNavigation(e) {
    // 키코드 변수 값 참조
    var key = e.keyCode;
    // 활성화된 탭 정보를 가져와 변수에 참조
    var $tab = $tabs.filter('.active');
    var index = getIndex.call($tab) + 1;
    var length = $tabs.length;
    // 사용자가 키보드 ←, ↑ 누를 경우 처리되는 조건
    if (key === 37 || key === 38) {
        // 유효한 인덱스 값을 설정
        index = --index > 0 ? index : length;
    }
    // 사용자가 키보드 →, ↓ 누를 경우 처리되는 조건
    else if (key === 39 || key === 40) {
        // 유효한 인덱스 값을 설정
        index = ++index <= length ? index : 1;
    }
    // 설정한 키보드 키가 아닌 경우 함수 종료
    else {
        return;
    }
    // 해당 인덱스의 탭 활성화
    activeTab(index);
    // 활성화 된 탭에 포커스 설정
    $tabs.filter('.active').focus();
}
```

작성한 키보드 방향키 내비게이션 프로그래밍이 정상 작동하는지 확인해보자. 탭키를 눌러 탭으로 포커스를 이동한 다음 키보드 방향키를 눌러보면 슬라이드가 정상 작동하고, 상태 또한 자동으로 변경되는 결과를 확인할 수 있다.

updateButtonText() 인디케이터 포커스 상태에서 좌우 방향키로 콘텐츠 탐색

키보드 인터랙션

구현된 캐럴셀 UI의 사용자 키보드 인터랙션을 정리해보면 다음과 같다.

키보드	인터랙션
좌우 방향키	인디케이터 이전/다음 항목을 탐색 및 포커스 한다.
탭키	인디케이터 〉 활성 콘텐츠 〉 이전/다음 버튼 순으로 탐색한다.

21. 콤보박스 (Combobox)

• 기존 코드의 문제점들

콤보박스(Combobox)는 옵션(Option)을 목록으로 제공하여 사용자가 선택할 수 있는 컴포넌트로, 웹 브라우저에 기본 내장된 내장된 셀렉트박스(〈select〉 요소)와 유사하다. 내장된 컴포넌트의 경우 접근성 이슈가 발생하지 않아 사용이 권장된다.

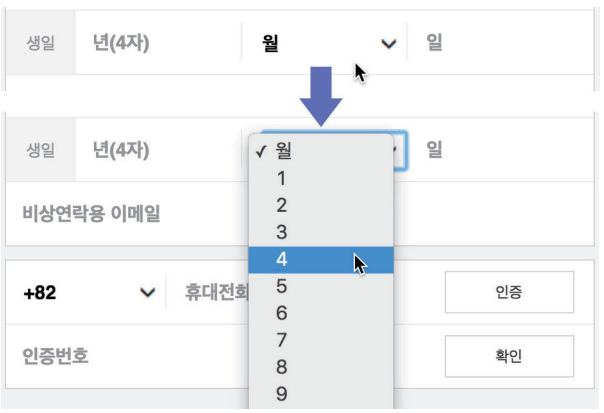
회원 가입 폼에 사용된 내장 컴포넌트 「셀렉트박스」

하지만 웹 브라우저에 내장된 컴포넌트 셀렉트의 경우, 플랫폼 또는 웹 브라우저 마다 각기 다른 디자인을 제공하고 있어, 이러한 웹 환경적 상황을 잘 모르는 디자이너와 협업 과정에서 때때로 의견 대립이 일어나기도 한다.

이러한 과정에서 디자이너와 협의 하에 적절한 선에서 타협이 이뤄지게 되고, 내장된 컴포넌트를 사용하되 디자인의 일부만 변경하는 선으로 결정하거나, 내장된 컴포넌트가 아닌 사용자 정의 컴포넌트를 제작하기도 한다.

다음에 등장하는 그림의 경우는 셀렉트박스의 겉면만 디자이너의 요구에 맞춰 비주얼 디자인을 적용하고, 사용자가 클릭한 다음 펼쳐지는 옵션 목록의 경우 내장 컴포넌트가 적용된 예이다.

비주얼 디자인의 일부만 변경한 내장 컴포넌트 「셀렉트박스」



반면 아래 그림의 경우는 사용자가 직접 정의한 컴포넌트로, 겉면뿐만 아니라 옵션 목록까지 모두 비주얼 디자인이 적용된 예이다.

비주얼 디자인 일관성을 목표로 하는 사용자 정의 컴포넌트「콤보박스」

콤보박스의 비주얼 디자인이 모든 웹 브라우저에서 일관적이길 바라는 디자이너의 바람을 반영하여 사용자 정의 컴포넌트를 제작하는 것은 분명 필요하다. 하지만 모양만 그럴듯하게 셀렉트박스와 유사해서는 안된다는 사실을 잊어서는 안된다. 내장된 컴포넌트인 셀렉트박스의 접근성과 기능까지 모방해야 한다.

아래 코드는 위의 사용자 정의 컴포넌트 마크업으로 문법적으로는 아무런 문제가 없다. 뿐만 아니라 비장애인에게 유려한 디자인을 뽐내는 멋진 컴포넌트로 보여질 수 있을 것이다.

그러나 스크린리더 사용자는 해당 컴포넌트가 콤보박스라는 사실을 전혀 알 길이 없다. 막연하게 나열된 목록을 오랫동안 살펴본 후에야 해당 기능에 대해 힘겹게 유추볼 수 있을 것이다.

비주얼 디자인 일관성을 목표로 하는 사용자 정의 컴포넌트「콤보박스」

```
<div class="selectbox_wrap type3">
    <input type="button" class="value_holder" value="SKT">
    <div class="new_selectbox">
        <ul>
            <li>SKT</li>
            <li>LGT</li>
            <li>KT</li>
            <li>기타통신사</li>
        </ul>
    </div>
</div>
```

• WAI-ARIA를 사용해야 하는 이유

사용자 정의 컴포넌트 접근성 향상에 WAI-ARIA가 큰 힘이 될 수 있다. 공들여 디자인 한 컴포넌트에 올바른 의미를 부여하고, 내장된 셀렉트박스에 비견되는 뛰어난 접근성을 가진 컴포넌트 제작이 가능해진다.

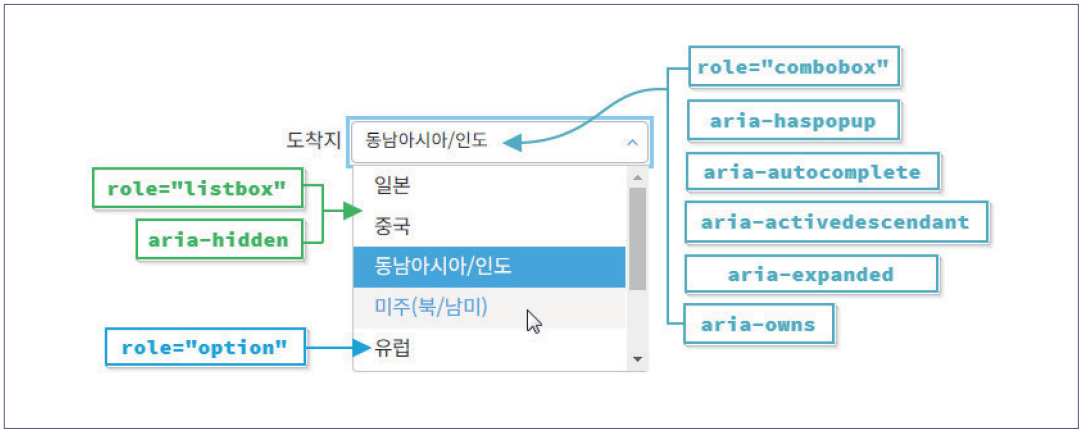
의미적으로는 사용자 정의 컴포넌트에 콤보박스(Combobox) 역할을, 콤보박스 목록에 해당되는 요소에는 리스트박스(Listbox)의 역할을, 그리고 리스트박스의 각 항목에는 옵션(Option) 역할을 부여한다.

기능적으로는 사용자의 선택에 따른 변경 값을 스크린리더에서 바로 읽어줄 수 있도록 할 수 있으며, 리스트박스의 펼침/접힘 상태 또한 안내해줄 수 있다.

• WAI-ARIA 전체적인 그림과 설명

콤보박스 컴포넌트 구현을 위해 의미적, 기능적 설정을 위한 WAI-ARIA 역할(Role), 속성(Property), 상태(State) 구조를 정리해보자. WAI-ARIA를 사용하여 설정하여야 할 것은 콤보박스와 리스트박스, 옵션이다. 적절한 컴포넌트 역할을 〈input〉 요소와 〈ul〉, 〈li〉 요소에 각각 부여하여 의미를 설정한다. 그리고 적용 가능한 속성/상태를 고려하여 추가하는 것으로 콤보박스 컴포넌트로 변모시킬 수 있다.

아이디 입력 필드에 WAI-ARIA를 사용하여 설명을 연결



• WAI-ARIA 속성 정리

앞서 살펴본 예제를 포함해 폼 컴포넌트에 적용 가능한 WAI-ARIA 속성을 정리해보자.

요소	역할	속성/상태	설명
〈input〉	combobox		콤보박스 역할
		aria-autocomplete	입력 완성 추천 제공 (list | item)
		aria-owns=""	리스트박스와 관계 형성 (부모/자식)
		aria-haspopup=""	콤보박스와 연관된 리스트박스 팝업 존재 유무 설정
		aria-readonly=""	읽기 전용 설정 (편집이 가능하지 않음)
		aria-expanded=""	펼침/접힘 상태를 제공
		aria-activedescendant=""	활성화된 자손 요소 정보를 실시간으로 제공
〈ul〉	listbox		리스트박스 역할
		aria-hidden=""	화면 표시/비표시 설정
〈li〉	option		옵션 역할

• 소스 코드 정리

사용자에게 제공될 콤보박스는 〈input〉 요소에 combobox 역할(Role)을 추가하는 것에서 시작한다. 이어서 사용자에게 입력 완성 추천이 목록으로 제공되는 컴포넌트라는 정보를 알려줄 수 있도록 aria-autocomplete 속성(Property)을 list로 설정한다.

그리고 입력 완성 추천 목록인 리스트박스가 하위 메뉴 형태로 존재함을 알려주기 위해 aria-haspopup 속성(Property)을 true로 설정한다. 현재 제작중인 콤보박스는 사용자의 편집이 가능하지 않으므로 편집이 가능하지 않음을 aria-readonly 속성을 true로 설정함으로 정보를 제공한다.

이어서 제작할 리스트박스를 콤보박스에 연결해야 하므로 aria-owns 속성에 리스트박스 id 속성 값을 설정하고, 기본적으로 리스트박스가 보이지 않는 접힘 상태가 될 수 있도록 aria-expanded 속성 값을 false로 초기 설정한다.

```
<!-- 콤보박스 레이블 -->
<label class="ui-combobox-label" for="cb-travel">도착지</label>
<!-- 콤보박스 역할 -->
<input
  type="text"
  role="combobox"
  readonly="true"
  id="cb-travel"
  class="ui-combobox"
  aria-autocomplete="list"
  aria-owns="cb-travel-list"
  aria-haspopup="true"
  aria-readonly="true"
  aria-expanded="false"
  aria-activedescendant=""
  placeholder="도착 지역 선택">
```

이어서 콤보박스의 버튼 역할을 수행할 〈button〉 요소를 추가한 다음, 내부에 아이콘 폰트를 삽입할 〈span〉 요소를 추가하고 해당 요소에 각각 aria-label, aria-hidden 속성을 설정한다. 버튼 텍스트 대신 레이블을 이용하여 적절한 내용을 읽을 수 있도록 제공하고, 아이콘 폰트의 경우 스크린리더에서 읽지 않도록 감춘다.

그리고 리스트박스 역할을 〈ul〉 요소에 부여하고, 내부의 〈li〉 요소에는 옵션 역할을 설정한다.

```
<!-- 콤보박스 버튼 -->
<button
  tabindex="-1"
  type="button"
  class="ui-combobox-button"
  aria-label="지역 선택 목록 열기" >
  <span class="fa fa-angle-down"  aria-hidden="true" ></span>
</button>
<!-- 콤보박스 리스트박스 -->
<ul
  id="cb-travel-list"
  class="ui-combobox-list"
  role="listbox"
  aria-hidden="true"
  tabindex="-1">
  <li
    id="cb-travel-option-01"
    class="ui-combobox-list-option"
    role="option">일본</li>
  <li
    id="cb-travel-option-02"
    class="ui-combobox-list-option"
    role="option" >중국</li>
  <li
    id="cb-travel-option-03"
    class="ui-combobox-list-option"
    role="option">동남아시아/인도</li>
  <li
    id="cb-travel-option-04"
    class="ui-combobox-list-option"
    role="option">미주(북/남미)</li>
  <li
    id="cb-travel-option-05"
    class="ui-combobox-list-option"
    role="option">유럽</li>
  <li
    id="cb-travel-option-06"
    class="ui-combobox-list-option"
```

```
    role="option">대양주/괌</li>
  <li
    id="cb-travel-option-07"
    class="ui-combobox-list-option"
    role="option">러시아/몽골/중앙아시아</li>
  <li
    id="cb-travel-option-08"
    class="ui-combobox-list-option"
    role="option">중동/아프리카</li>
</ul>
```

이어서 작성되는 코드는 자바스크립트(jQuery 활용) 파트이다. 먼저 키보드 키를 손쉽게 식별하기 위한 키보드 객체를 생성한다. 콤보박스 컴포넌트에서 사용되는 키를 각각 포함해야 한다.

```
// 키보드 객체
var keys = {
  'tab': 9,
  'enter': 13,
  'alt': 18,
  'esc': 27,
  'space': 32,
  'pageup': 33,
  'pagedown': 34,
  'left': 37,
  'up': 38,
  'right': 39,
  'down': 40,
};
```

그리고 컴포넌트를 초기화하는 코드를 추가한다. 콤보박스, 콤보박스 버튼, 리스트박스, 리스트박스 옵션에 해당되는 요소를 각기 참조하는 변수와 컴포넌트 초기 변수를 정의한다. 모든 변수 정의를 마무리한 다음 이벤트를 설정할 함수를 실행시킨다.

```
// 초기화
// 문서 객체 참조 → jQuery 객체화
var $combobox_wrapper = $('.ui-combobox-wrapper');
var $combobox = $combobox_wrapper.find('.ui-combobox');
var $combobox_button = $combobox_wrapper.find('.ui-combobox-button');
var $combobox_list = $combobox_wrapper.find('.ui-combobox-list');
var $combobox_options = $combobox_list.find('.ui-combobox-list-option');
// 기본 값 변수 초기화
var options_len = $combobox_options.length;
var list_height = $combobox_list.outerHeight();
var input_default_content = '도착 지역 선택';
var selected_option_idx = 0;
var focused_option = null;
// 이벤트 바인딩
bindEventHandler();
```

이벤트 설정 함수에서는 콤보박스, 콤보박스 버튼, 리스트박스 옵션에 각각 수행해야 할 이벤트에 핸들러(함수)를 연결한다.

```
// bindEventHandler //
function bindEventHandler() {
  // input
  $combobox.on({
    'click': inputClick,
    'keydown': inputKeydown,
    'blur': inputBlur
  });
  // button
  $combobox_button.on({
    'click': toggleList,
    'keydown': inputKeydown
  });
  // option
  $.each($combobox_options, function(idx, el) {
    var $option = $combobox_options.eq(idx);
    $option.on('click', $.proxy(optionsClick, $option, idx));
  });
}
```

콤보박스에 해당하는 〈input〉 요소의 이벤트 핸들러를 각각 정의한다. 아래 정의된 함수 코드를 살펴보며 주석을 통해 코드를 이해해보자.

```
// input 이벤트 함수 //
// 인풋 클릭 함수
function inputClick(e) {
  // 콤보 리스트가 닫혀있다면, 열음
  !isListOpen() && openList();
}
// 인풋 키다운 함수
function inputKeydown(e) {
  // alt 키를 눌렀다면_
  if (e.altKey) {
    // ↑ 키 누르면, 메뉴 열림
    if (e.keyCode === keys.up) {
      openList();
      // 콤보 리스트를 띄우면서 현재 활성화된 콤보박스 값을 저장
      $combobox.data('memory-value', $combobox.val());
    }
    // ↓ 키 누르면, 메뉴 닫힘
    e.keyCode === keys.down && closeList();
    return; // 종료
  }
  // 콤보 리스트가 열려있다면_
  if (isListOpen()) {
    // ←, ↑ 키를 누르면, 이전 옵션 탐색
    e.keyCode === keys.up && prevSelectOption();
    // →, ↓ 키를 누르면, 다음 옵션 탐색
    e.keyCode === keys.down && nextSelectOption();
    // pageUp 키 누르면, 처음 옵션 이동
    e.keyCode === keys.pageup && firstSelectOption();
    // pageDown 키 누르면, 마지막 옵션 이동
    e.keyCode === keys.pagedown && lastSelectOption();
    // Esc 키를 누르면, 취소
    e.keyCode === keys.esc && escapeSelectOption();
    // Enter, Space 키를 누르면, 활성화
    (e.keyCode === keys.enter || e.keyCode === keys.space) && activeSelectOp-
tion();
    return; // 종료
  }
```

```
  // 콤보 리스트가 열려있지 않다면_
  else {
    // ↑ 키 누르면, 이전 옵션 선택
    if (e.keyCode === keys.up) {
      prevSelectOption();
      activeSelectOption();
    }
    // ↓ 키 누르면, 다음 옵션 선택
    if (e.keyCode === keys.down) {
      nextSelectOption();
      activeSelectOption();
    }
  }
}
// input 블러 함수
function inputBlur(e) {
  //  0.1초 뒤에 콤보 리스트 닫음
  global.setTimeout(closeList, 100);
}
// input 포커스 > 선택 함수
function focusSelectInput() {
  $combobox.focus();
}
```

이어스 콤보박스에 연결된 리스트박스를 열고 닫는 함수를 정의해본다.

```
// 콤보 리스트 함수 //
// 콤보 리스트 열림/닫힘 여부 반환 함수
function isListOpen() {
  return $combobox.attr('aria-expanded') === 'true' ? true : false;
}
// 콤보 리스트 토글 함수
function toggleList() {
  isListOpen() ? closeList() : openList();
}
// 콤보 리스트 열기 함수
function openList() {
  $combobox.attr('aria-expanded', true);
  $combobox_list.attr('aria-hidden', false);
```

```
  $combobox_button
    .attr('aria-label', '지역 선택 목록 닫기')
    .find('span').removeClass('fa-angle-down').addClass('fa-angle-up');
}
// 콤보 리스트 닫기 함수
function closeList() {
  $combobox.attr('aria-expanded', false);
  $combobox_list.attr('aria-hidden', true);
  $combobox_button
    .attr('aria-label', '지역 선택 목록 열기')
    .find('span').removeClass('fa-angle-up').addClass('fa-angle-down');
}
```

리스트 옵션을 클릭할 때 처리되는 함수와 각 옵션으로 이동하는 탐색 함수를 추가로 작성한다.

```
// 콤보 옵션 함수 //
// 옵션 클릭 함수
function optionsClick(idx, e) {
  // 선택된 옵션 .selected 라디오 클래스 적용
  radioOption(idx);
  activeSelectOption();
}
// 처음 옵션 선택 함수
function firstSelectOption() {
  radioOption(0);
}
// 마지막 옵션 선택 함수
function lastSelectOption() {
  radioOption(options_len - 1);
}
// 이전 옵션 탐색 함수
function prevSelectOption() {
  radioOption(--selected_option_idx < 0 ? options_len - 1 : selected_option_
idx);
}
// 다음 옵션 탐색 함수
function nextSelectOption() {
  radioOption(++selected_option_idx > options_len - 1 ? 0 : selected_option_
idx);
```

```
// 다음 옵션 탐색 함수
function nextSelectOption() {
  radioOption(++selected_option_idx > options_len - 1 ? 0 : selected_option_
idx);
}
// 옵션 라디오 클래스/선택
function radioOption(idx) {
  // 선택된 옵션 인덱스 업데이트
  var $option = $combobox_options.eq(idx);
  focused_option = $option;
  selected_option_idx = idx;
  $combobox_options.filter('.selected').removeClass('selected');
  $option.addClass('selected');
  updateSelectedOption(focused_option.attr('id'));
  updateInput(focused_option.text());
  updateListScrollTop($option.position().top);
}
// 포커스된 옵션 활성화 함수
function activeSelectOption() {
  updateInput(focused_option.text());
  isListOpen() && closeList();
  focusSelectInput();
}
// Esc 키를 누를 경우 실행되는 취소 함수
function escapeSelectOption() {
  var memory_value = $combobox.data('memory-value');
  var id = $combobox_options.filter(':contains(' + memory_value + ')').at-
tr('id');
  updateInput(memory_value);
  updateSelectedOption(id);
  isListOpen() && closeList();
  focusSelectInput();
}
```

마무리로 이벤트에 따라 상태가 변경되도록 하는 업데이트 함수를 정의한다.

```
// 업데이트 //
// input 값 업데이트 함수
function updateInput(value) {
  $combobox.val(value);
}
// 선택된 옵션 id 값 업데이트 함수
function updateSelectedOption(id) {
  $combobox.attr('aria-activedescendant', id);
}
// 콤보 리스트 스크롤 높이 업데이트 함수
function updateListScrollTop(top) {
  $combobox_list.stop().animate({ 'scrollTop': top }, 200, 'linear');
}
```

• 키보드 인터랙션

키보드	인터랙션
위쪽 방향키	이전 옵션으로 이동
아래쪽 방향키	다음 옵션으로 이동
Alt + 위쪽 방향키	리스트박스 열림
Alt + 아래쪽 방향키	리스트박스 닫힘
Page Up	첫번째 옵션으로 이동
Page Down	마지막 옵션으로 이동
Esc키	실행 취소 (리스트 닫힘, 콤보박스로 포커스 이동)
엔터키	실행 (리스트 닫힘, 콤보박스로 포커스 이동)

예제로 살펴보는 WAI-ARIA

발 행 인 서병조
발 행 일 2016.11(비매품)
발 행 처 한국정보화진흥원
주 소 41068 대구광역시 동구 첨단로 53(신서동)
홈페이지 www.nia.or.kr | www.wah.or.kr

기 획 김정태 (미래창조과학부 팀장)
손창용 (미래창조과학부 사무관)
김봉섭 (한국정보화진흥원 팀장)
한정기 (한국정보화진흥원 수석)

집 필 성영한 (HBI기술연구소)
김데레사 (멀티캠퍼스)
김봉주 (한진정보통신)
지훈 (UALab)
지성봉 (콘텐츠연합플랫폼)

감 수 김혜일

• 본 책자는 아래 홈페이지에서 파일로 다운 받으실 수 있습니다.
웹 접근성 연구소 홈페이지(www.wah.or.kr)

• 본 책자에 대한 문의는 한국정보화진흥원(jhan@nia.or.kr)으로 연락하여 주시기 바랍니다.

예제로 살펴보는

WAI-ARIA

미래창조과학부

NIA 한국정보화진흥원