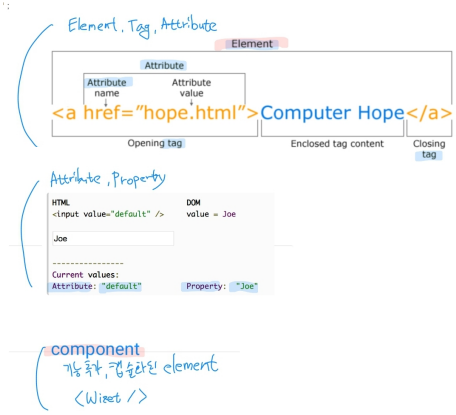
WebJS

Contents

Chapter 1. 관계

- **Node**
  - Document
    - HTMLDocument
  - CharacterData
    - Text
      - Comment
  - Element
    - HTMLElement
      - HTMLHeadElement
      - HTMLLIElement
  - Attr

Element, Tag, Attribute, Property, Component



Chapter 2. BOM (Browser Object Model)

navigator, screen, location, frames, history, XMLHttpRequest …

2.1. Windows 객체

- Window 객체는 모든 객체가 소속된 객체이고, 전역객체이면서, 창이나 프레임을 의미
- 전역변수와 함수가 사실은 window 객체의 프로퍼티와 메소드
- 모든 객체는 사실 window의 자식

```
<script>
windows.alert('a')
alert('a') // = windows.alert('a')
</script>
```

2.1.1. Communication

- alert: 경고창

```
<input type="button" value="alert" onclick="alert('hello world');" />
```

- confirm: 확인창

```
<input type="button" value="confirm" onclick="func_confirm()" />
<script>
function func_confirm(){
  if(confirm('ok?')) { alert('ok') }
  else { alert('cancel') }
}
</script>
```

- prompt: 팝업 입력창

```
<input type="button" value="prompt" onclick="func_prompt()" />
<script>
function func_prompt(){
  if(prompt('id?') === 'egoing'){ alert('welcome') }
  else { alert('fail') }
}
</script>
```

2.1.2. Location

문서 주소 제공

- URL

```
console.log(location.protocol, location.host, location.port, location.pathname, location.search,
location.hash)
```

- reload

```
location.reload();
```

2.1.3. Navigator

브라우저 정보 제공

2.1.4. 창 제어

새 창 열기: window.open

Chapter 3. DOM (Document Object Model)

document, tag, class, id, element, component, attribute, property …

3.1. 제어 대상 찾기

인자로 전달된 tag, class, id 에 해당하는 객체들을 찾아서 그 리스트를 NodeList라는 유사 배열에 담아서 반환, 중첩하여 좀힐수 있음

- document.getElementsByTagName('string'): 태그로 찾기
- document.getElementsByClassName('string'): 클래스로 찾기
- document.getElementsById('string'): 아이디로 찾기
- document.getElementById('string'): 아이디로 1개 찾기
- document.querySelectorAll('string'): css 선택자로 찾기
- document.querySelector('string'): css 선택자로 1개 찾기

```
<div id = 'iA' class = 'cA'> <p id = 'iB' class = 'cB'> </p> </div>

<script>
let getList = document.getElementsByTagName('div').getElementsByTagName('p');
let getList = document.getElementsByClassName('cA').getElementsByClassName('cB');
let getList = document.getElementsById('iA').getElementsById('iB');
let getList = document.querySelectorAll('div').querySelectorAll('.cB');
getList[0].style.color='red'

let getItem = document.getElementById('iA') // document.getElementById('iA') = iA, 중첩은 불가
(iA.iB... X)
let getItem = document.querySelector('div').style.color='red';
let getItem = document.querySelector('.cA').style.color='red';
getItem.style.color='red'
</script>
```

3.2. Element 객체

3.2.1. 식별

- Element.tagName: 태그 이름 조회 가능. 변경 불가.

- Element.id: 문서에서 id는 단 하나만 등장할 수 있는 식별자. id 조회, 변경 가능.
- Element.className: 클래스는 여러개의 엘리먼트를 그룹핑할 때 사용. 조회, 변경, 추가 가능
- Element.classList: 클래스 조회, 추가, 제거, 토글 (없으면 추가, 있으면 제거) 가능

```
<li id="iA" class="cA">Text</li>

<script>
console.log(iA.tagName); // li

console.log(iA.id);
iA.id = 'iB';

console.log(iA.classList); // ['cA', value: 'cA']
iA.classList.add('cB'); // 추가
iA.classList.remove('cB'); // 제거
iA.classList.toggle('cB'); // 토글 (추가 시 true, 제거 시 false 반환)

</script>
```

3.2.2. 조회

- Element.getElementsByClassName('string')
- Element.getElementsByTagName('string')
- Element.querySelector('string')
- Element.querySelectorAll('string')

3.2.3. Attribute

Attribute는 HTML에서 태그명만으로는 부족한 부가적인 정보

- Element.getAttribute(name)
- Element.setAttribute(name, value)
- Element.hasAttribute(name)
- Element.removeAttribute(name)

```
<a id="iA" href="http://google.com">Google</a>

<script>
iA.getAttribute('href'); // 조회, http://google.com
iA.setAttribute('title', 'Google'); // 설정, title="Google"
iA.removeAttribute('title', 'Google'); // 삭제
iA.hasAttribute('title') // 확인, 있으면 true, 없으면 false 반환
</script>
```

Attribute vs. Property

Attribute 방식인 setAttribute('class', 'important')와 Property 방식인 className = 'important'는 같은 결과를 만든다.

```
iA.setAttribute('title', 'Google') // Attribute
iA.title = 'Google' // Property
```

Property 방식은 좀 더 간편하고 속도도 빠르지만 자바스크립트의 이름 규칙 때문에 실제 html 속성의 이름과 다른 이름을 갖는 경우가 있다.
Attribute 방식은 titleA 와 같은 임의의 형식을 추가할 수 있으나 Property 방식은 미리 정해진 형식만 추가할 수 있다.

Attribute	Property
readonly	readOnly
rowspan	rowSpan
colspan	colSpan
usemap	userMap
frameborder	frameBorder
for	htmlFor
maxlength	maxLength

3.3. Node 객체

모든 DOM 객체는 Node 객체를 상속 받는다.

3.3.1. 관계

노드간 연결 정보를 조회하여 문서를 프로그래밍적으로 탐색

- Node.childNodes: 자식 노드들을 유사배열에 담아 리턴
- Node.firstChild: 첫번째 자식 노드 (#text = 태그사이에 공백이나 줄바꿈이 있으면 공백이 자식엘리먼트로 잡힘)

#text 무시 API

- firstElementChild
- lastElementChild
- nextElementSibling
- previousElementSibling

- Node.lastChild: 마지막 자식 노드
- Node.nextSibling: 다음 형제 노드
- Node.previousSibling: 이전 형제 노드
- Node.contains():
- Node.hasChildNodes():

```
iA.childNodes // [text, h1, text, div#toc.toc2, text]
iA.firstChild // #text
iA.lastChild // #text
iA.nextSibling // #text
iA.previousSibling // #text

iA.firstChild.nextSibling // <h1>WebJS</h1>
```

3.3.2. 식별

각 구성요소의 소속 카테고리 확인

- Node.nodeType: 노드 타입을 고유의 숫자로 반환
- Node.nodeName: 노드 태그를 string으로 반환

```
iA.nodeType // 1
iA.firstChild.nodeType // 3

iA.nodeName // 'div'
```

3.3.3. 값

Node 객체의 값 확인

- Node.nodeValue
- Node.textContent

3.3.4. 자식관리

Node 객체의 자식 (Child) 추가 또는 제거

- Node.appendChild(newElement): 노드의 마지막 자식으로 newElement 추가 후 추가한 newElement 반환
- Node.insertBefore(newElement, refElement): newElement 를 refElement 앞에 추가 후 후 추가한 newElement 반환
- Node.removeChild(targetElement): targetElement 를 제거 후 제거한 targetElement 반환
- Node.replaceChild(newElement, oldElement): oldElement 를 newElement 로 교체 후 oldElement 를 반환

```
iA.appendChild(document.createElement('div'))
iA.insertBefore(document.createElement('div'), iB)
iA.removeChild(iB)
iA.replaceChild(document.createElement('div'), iB)
```

3.3.5. 제어

- Node.innerHTML: 문자열로 자식 노드를 변경 (=) 하거나 추가 (+=) 하고 변경된 자식 노드를 반환하거나, 자식 노드의 값 조회
- Node.outerHTML: 문자열로 자신을 포함한 노드를 변경 (=) 하거나 추가 (+=) 하고 변경된 자신을 포함한 노드를 반환하거나, 자신을 포함한 노드의 값 조회
- Node.innerText: innerHTML 노드 생성 방식은 같으나, html 코드를 제외한 문자열만 조회
- Node.outerText: outerHTML 노드 생성 방식은 같으나, html 코드를 제외한 문자열만 조회

```
iA.innerHTML = '<div id = iB> B' // <div id="iB"> B </div> 미완성 문법 자동 보완
iA.innerHTML // <div id = iB> ... </div>

iA.outerHTML = '<div id = iB> B'
iA.outerHTML // <div id = iA> ... </div>

iA.innerText // ...
iA.outerText // ...
```

- Node.insertAdjacentHTML(position: string, string): 특정 위치에 string 으로 작성된 노드를 추가

  position

  - 'beforebegin': element 앞에
  - 'afterbegin': element 안에 가장 첫번째 child
  - 'beforeend': element 안에 가장 마지막 child
  - 'afterend': element 뒤에

```
<script>
iA.insertAdjacentHTML('beforebegin', '<div> ... </div>')
</script>
<!-- beforebegin -->
<p>
<!-- afterbegin -->
foo
<!-- beforeend -->
</p>
<!-- afterend -->
```

NOTE 유사 API: insertAdjacentElement, insertAdjacentText

Chapter 4. Document 객체

새로운 노드를 생성해주는 역할. 노드 변경 API와 동일. Node 객체 참조.

Chapter 5. Text 객체

CharacterData를 상속. 텍스트 객체는 텍스트 노드에 대한 DOM 객체

5.1. 값 조회

- CharacterData.data: 텍스트 노드의 텍스트 데이터를 반환
- CharacterData.nodeValue: data와 같지만 속성 및 주석 노드 함께 반환
- CharacterData.textContent: 해당 노드와 그 자손의 텍스트 데이터를 모두 반환
- CharacterData.wholeText: ?

```
iA.firstChild.data
iA.firstChild.nodeValue
iA.firstChild.textContent
iA.firstChild.wholeText
```

5.2. 값 제어

- appendData(string): 텍스트 노드의 끝에 문자열 추가
- deleteData(offset: number, count: number): offset (0 부터) 지점부터 count 갯수 (byte) 만큼 문자 삭제
- insertData(offset: number, data: string): offset (0 부터) 지점에 data 추가
- replaceData(offset: number, count: number, data: string): offset (0 부터) 지점부터 count 갯수 (byte) 만큼 문자를 삭제하고 data 추가
- substringData(offset: number, count: number): offset (0 부터) 지점부터 count 갯수 (byte) 만큼 문자를 리턴

Chapter 6. Event

이벤트 (event)는 변화가 생기는 사건을 의미. 브라우저에서의 사건이란 사용자가 클릭을 했을 '때', 스크롤을 했을 '때', 필드의 내용을 바꾸었을 '때'와 같은 것

Event target

이벤트가 일어날 객체

- event.target: 부모 요소의 핸들러는 이벤트가 정확히 어디서 발생했는지 등에 대한 자세한 정보를 얻을 수 있음. 이벤트가 발생한 가장 안쪽의 요소는 타깃 (target) 요소라고 불리고, event.target을 사용해 접근 가능. 버블링이 진행되어도 변경되지 않음.
- this (= event.currentTarget): this는 '현재' 요소로, 현재 실행 중인 핸들러가 현재 할당된 요소를 참조.

캡처링, 버블링과 관련된 내용은 전파 항목 확인

Event type

이벤트의 종류. 이미 정해져 있음.

- 전체 이벤트 목록: https://developer.mozilla.org/en-US/docs/Web/Events

Event handler

이벤트가 발생했을 때 동작하는 코드

6.1. 등록 방법

Inline

```
<input type="button" id="target" onclick="alert('Hello world');" value="버튼" />
<input type="button" onclick="alert('Hello world');" value="버튼" />
```

Property Listener

```
<input type="button" id="iA" value="again" />

<script>
  iA.onclick = function(event){
  alert('Hello world, '+ event.target.value)
  }
</script>
```

addEventListener(event:string, eventHandler)

여러개의 이벤트 핸들러 등록 가능, eventType 의 이벤트 발생 시, eventHandler 실행

```
<input type="button" id="iA" value="버튼" />

<script>
iA.addEventListener('click', function(event){
  alert(event.target.id)});
iA.addEventListener('click', function(event){
  alert(event.target.value)});
</script>
```

6.2. 전파

- 버블링

  한 요소에 이벤트가 발생하면, 이 요소에 할당된 핸들러가 동작하고, 이어서 부모 요소의 핸들러가 동작. 가장 최상단의 조상 요소를 만날 때까지 이 과정이 반복되면서 요소 각각에 할당된 핸들러가 동작.

  - 버블링 중단

    이벤트 객체의 메서드인 event.stopPropagation()를 사용하여 핸들러에게 이벤트를 처리하고 난 후 버블링을 중단하도록 명령 가능.

    ```
    <body onclick="alert(`버블링은 여기까지 도달하지 못합니다.`)">
      <button onclick="event.stopPropagation()">클릭해 주세요.</button>
    </body>
    ```

    NOTE

    이벤트 버블링을 막아야 하는 경우는 거의 없습니다. 버블링을 막아야 해결되는 문제라면 커스텀 이벤트 등을 사용해 문제를 해결할 수 있습니다.

- 캡처링

  이벤트가 최상위 조상에서 시작해 아래로 전파되고 (캡처링 단계), 이벤트가 타깃 요소에 도착해 실행된 후 (타깃 단계), 다시 위로 전파됩니다 (버블링 단계). 이런 과정을 통해 요소에 할당된 이벤트 핸들러를 호출

  버블링과 캡처링은 '이벤트 위임 (event delegation)'의 토대가 됩니다. 이벤트 위임은 강력한 이벤트 핸들링 패턴입니다.

- 이벤트 위임

  이벤트 위임은 비슷한 방식으로 여러 요소를 다뤄야 할 때 사용됩니다. 이벤트 위임을 사용하면 요소마다 핸들러를 할당하지 않고, 요소의 공통 조상에 이벤트 핸들러를 단 하나만 할당해도 여러 요소를 한꺼번에 다룰 수 있습니다.

  공통 조상에 할당한 핸들러에서 event.target을 이용하면 실제 어디서 이벤트가 발생했는지 알 수 있습니다. 이를 이용해 이벤트를 핸들링합니다.

  https://ko.javascript.info/event-delegation

6.3. 기본 동작의 취소

웹브라우저의 구성요소들의 정해진 기본적인 동작 방법 (기본 이벤트) 사용자가 만든 이벤트를 이용해서 취소

- Inline: 이벤트 리턴값이 false 이면 기본 동작 취소
  return false
- property: 이벤트 리턴값이 false 이면 기본 동작 취소
  return false
- addEventListener(): 이벤트 객체의 preventDefault() 메소드를 실행하면 기본 동작 취소
  event.preventDefault()

```
iA.addEventListener('click', event => {
  dosomething();
  event.preventDefault()
} )
```

6.4. 이벤트 타입

- submit: form의 정보를 서버로 전송하는 명령인 submit시에 일어나는 이벤트.

```
<input type="submit" />
<script>
iA.addEventListener('submit', e => {...})
</script>
```

- change: 폼의 값이 변경 되었을 때 발생

```
<input />
<script>
iA.addEventListener('change', e => {...})
</script>
```

- focus, blur: focus는 엘리먼트에 포커스가 생겼을 때, blur은 포커스가 사라졌을 때 발생

```
iA.addEventListener('blur', e => {...})
iA.addEventListener('focus', e => {...})
```

NOTE

blur, focus 발생 제외 태그: <base>, <bdo>,
, <head>, <html>, <iframe>, <meta>, <param>, <script>, <style>, <title>

- DOMContentLoaded, load: DOMContentLoaded는 문서에서 스크립트 작업을 할 수 있을 때 실행, load는 문서내의 모든 리소스(이미지, 스크립트)의 다운로드가 끝난 후 실행

```
window.addEventListener('DOMContentLoaded', e => {...} )
```

```
window.addEventListener('load', e => {...} )
```

- 마우스
  - click: 클릭했을 때 발생
  - dblclick: 더블클릭을 했을 때 발생
  - mousedown: 마우스를 누를 때 발생
  - mouseup: 마우스 버튼을 땔 때 발생
  - mousemove: 마우스를 움직일 때
  - mouseover: 마우스가 엘리먼트에 진입할 때 발생
  - mouseout: 마우스가 엘리먼트에서 빠져나갈 때 발생
  - contextmenu: 컨텍스트 메뉴 (팝업) 가 실행될 때 발생
  - event.clientX, event.clientY: 포인터 위치
- 키보드 조합: 특수키(alt, ctrl, shift)가 눌려진 상태를 감지
  - event.shiftKey
  - event.altKey
  - event.ctrlKey

Chapter 7. 네트워크 통신

7.1. Ajax

7.2. JSON

Appendix A: 주요 태그 종류

A.1. form

입력된 데이터를 서버로 전송. 태그는 전체 양식을 의미하며, 화면에 보이지 않는 추상적인 태그. input 태그 등으로 내부를 구성

Attribute

- name: 폼의 이름
- action: 폼 데이터가 전송되는 백엔드 url
- method: 폼 전송 방식 (GET / POST)
- …

```
<form name="testInputName" action="xxxx.xxx.xxx" method="POST">
<p> <strong>아이디</strong>
<input type="text" name="name" value="아이디 입력"> </p>
<p> <strong>비밀번호</strong>
<input type="password" name="password" value="비밀번호 입력"> </p>
<p> <strong>성별</strong>
<input type="radio" name="gender" value="M">남자
<input type="radio" name="gender" value="F">여자 </p>
<p> <strong>응시분야</strong>
<input type="checkbox" name="part" value="eng">영어
<input type="checkbox" name="part" value="math">수학 </p>
<p> <input type="submit" value="제출"> </p>
</form>
```

A.2. input

사용자 입력을 받음

- **type**
  - text: 일반 문자
  - password: 비밀번호
  - button: 버튼
  - submit: 양식 제출용 버튼
  - reset: 양식 초기화용 버튼

- radio: 한개만 선택할 수 있는 컴포넌트
- checkbox: 다수를 선택할 수 있는 컴포넌트
- file: 파일 업로드
- hidden: 사용자에게 보이지 않는 숨은 요소

A.3. select, option

드롭 다운 리스트 생성

```
<select>
<option value="ktx">KTX</option>
<option value="itx">ITX 새마을</option>
<option value="mugung">무궁화호</option>
</select>
```

Appendix B: CSS

B.1. CSS Selector

IMPORTANT

출처: https://code.tutsplus.com/ko/tutorials/the-30-css-selectors-you-must-memorize—net-16048

B.1.1. *

```
* {
 margin: 0;
 padding: 0;
}
```

고급 선택자로 이동하기 전에 초보자를 위해 쉬운 선택자부터 알아보죠.
별표는 페이지에 있는 전체 요소를 대상으로 합니다. 많은 개발자가 margin과 padding 값을 0으로 세팅하려고 이 선택자를 사용합니다. 간단한 테스트 용도로서는 괜찮습니다. 그러나, 저는 여러분에게 이 별표를 실전에서 사용하지 말라고 권합니다. 브라우저에 과부하가 걸리고, 사용하기에 적절하지 않습니다.

*를 자식 선택자에도 사용할 수 있습니다.

```
#container * {
 border: 1px solid black;
}
이 코드는 #container div의 자식 요소 전체를 대상으로 합니다. 한 번 더 말하지만, 이 선택자를
과다하게 사용하지 마세요.
```

B.1.2. #X

```
#container {
   width: 960px;
   margin: auto;
}
```

id 선택자. 선택자 앞에 해시(#) 기호를 붙여서 id를 대상으로 삼습니다. 가장 흔하고 쉽게 사용됩니다. 하지만, id 선택자를

사용할 때는 조심스러워야 합니다.
자문해 보세요. 이 요소를 대상으로 하기 위해 id를 필히 적용해야 할까요?
id 선택자는 유연성이 없고 재활용할 수 없습니다. 가능한 처음에 태그 명이나 새로운 HTML 요소 중 하나, 아니면 가상 클래스라도 적어 보세요.

B.1.3. .X

```
.error {
  color: red;
}
```

class 선택자입니다. id와 class의 차이점이라면, 후자는 여러 개의 요소를 대상으로 정할 수 있습니다. 스타일을 한 그룹의 요소에 적용할 때는 class를 사용하세요. 찾을 가망성이 거의 없는 요소에 id를 사용하고 그 유일한 요소에만 스타일을 적용하세요.

B.1.4. X Y

```
li a {
  text-decoration: none;
}
```

다음으로 가장 많이 언급하는 선택자는 descendant (하위 선택자) 입니다. 선택자를 이용해 더 상세히 작업해야 할 때, 이 선택자를 사용합니다. 가령, 전체 앵커 태그를 대상으로 하기보다 순서를 매기지 않는 목록 (unordered list) 에 있는 앵커만 대상으로 한다면 어떨까요? 하위 선택자를 사용하면 상세해집니다.

NOTE

꿀팁 - 선택자가 X Y Z A B.error처럼 보이면 여러분은 작업을 잘못하고 있습니다. 모든 요소에 꼭 가중치를 둬야 하는지를 늘 자문하세요.

B.1.5. X

```
a { color: red; }
ul { margin-left: 0; }
```

만일 여러분이 id나 class가 아닌 type에 따라 한 페이지에 있는 모든 요소를 대상으로 삼고 싶다면 어떨까요? 간단히 type 선택자를 이용하세요. 순서가 정해지지 않은 목록 전부를 대상으로 해야 한다면 ul {}를 쓰세요.

B.1.6. X:visited 와 X:link

```
a:link { color: red; }
a:visted { color: purple; }
```

클릭하기 전 상태의 앵커 태그 전체를 대상으로 하려고 :link 가상 클래스를 사용합니다.
:visited 가상 클래스로 하기도 합니다. 예상하듯이 이는 클릭했었거나 방문했던 페이지에 있는 앵커 태그에만 특정한 스타일을 적용할 수 있습니다.

B.1.7. X + Y

```
ul + p {
   color: red;
}
```

인접 선택자로 부르는 선택자입니다. 앞의 요소 **바로 뒤**에 있는 요소만 선택합니다. 위 코드에서 각 ul 뒤에 오는 첫 번째 단락의 텍스트만 빨간색이 됩니다.

B.1.8. X > Y

```
div#container > ul {
  border: 1px solid black;
}
```

일반 X Y와 X > Y의 차이점은 후자가 직계 자식만을 선택한다는 것입니다. 가령, 아래 마크업을 생각해 보세요.

```
<div id="container">
   <ul> // selected
      <li> List Item
        <ul> // not selected
           <li> Child </li>
        </ul>
      </li>
      <li> List Item </li>
   </ul>
</div>
```

#container > ul 선택자는 id가 container인 div의 **직계 첫번째 자손**인 ul만 대상으로 삼습니다. 예를 들어 첫 번째 li의 자식인 ul은 대상이 되지 않습니다.
이런 이유로 자식 선택자를 이용해 성능을 향상할 수 있습니다. 사실, 자바스크립트를 기반으로 하는 CSS 선택자 엔진으로 작업할 때 추천합니다.

B.1.9. X ~ Y

ul ~ p {

color: red;
}
이 형제 선택자는 X + Y와 유사하지만 덜 엄격합니다. 인접 선택자(ul + p)는 앞의 선택자 바로 뒤에 오는 첫 번째 요소만을 선택하지만, 이 선택자는 좀 더 관대합니다. 위의 예를 보면, ul 아래 있는 모든 p 요소를 선택할 것입니다.

데모 보기

호환성
IE7+
파이어폭스
크롬
사파리
오페라
10. X[title]
a[title] {
color: green;
}
속성 선택자(attributes selector)라고 말하며, 앞의 예에서 title 속성이 있는 앵커 태그만을 선택합니다. title이 없는 앵커 태그에는 특정한 스타일이 적용되지 않습니다. 그런데 더 상세히 작업해야 한다면 어떨까요? 음…

데모 보기

호환성
IE7+
파이어폭스
크롬
사파리
오페라
11. X[href="foo"]
a[href="https://net.tutsplus.com"] {
color: #1f6053; /* nettuts green */
}
위의 코드는 https://net.tutsplus.com;로 연결된 전체 앵커 태그에 스타일을 적용할 것입니다. 우리 브랜드 컬러인 녹색이 적용되겠지요. 그 외의 앵커 태그는 스타일의 영향을 받지 않습니다.

값을 큰따옴표로 감쌌음을 기억하세요. 자바스크립트 CSS 선택자 엔진을 사용할 때 활용하는 것도 잊지 마세요. 가능하면, 비공식적인 선택자보다 CSS3 선택자를 항상 사용하세요.

동작은 잘하겠지만, 융통성은 낮습니다. 만약 링크가 Nettuts+로 직접 이어지지만, 경로를 전체 url이 아닌 nettuts.com으로 한다면 어떨까요? 그 경우에 우리는 정규식 표현 문장을 약간 사용할 수 있습니다.

데모 보기

호환성
IE7+
파이어폭스
크롬
사파리
오페라
12. X[href*="nettuts"]
a[href*="tuts"] {
color: #1f6053; /* nettuts green */
}
야아. 우리에게 필요한 선택자네요. 별표는 입력값이 속성값 안 어딘가에 보여야 한다는 것을 표시합니다. 그렇게 이 구문은 nettuts.com, net.tutsplus.com 그리고 tutsplus.com까지도 적용하고 있습니다.

폭넓은 표현이라는 것을 알아 두세요. 만약 앵커 태그의 url에 tuts 문자열이 일부 Evato가 아닌 사이트로 연결되어 있다면 어떨까요? 더 자세히 작성해야 한다면, 문자열의 앞과 뒤에 ^와 $를 붙이세요.

데모 보기

호환성
IE7+
파이어폭스
크롬
사파리
오페라
13. X[href^="http"]
a[href^="http"] {
background: url(path/to/external/icon.png) no-repeat;
padding-left: 10px;
}
웹사이트에서 외부로 연결된 링크 옆에 작은 아이콘을 어떻게 보이게 했는지 궁금해한 적이 있나요? 틀림없이 전에 본 적이 있을 것입니다. 링크를 클릭하면 전혀 다른 웹사이트로 이동하리라는 것을 알게 해주니까요.

캐럿 기호를 이용하는 쉬운 작업입니다. 문자열의 시작을 표기하는 정규 표현식에서 흔히 사용됩니다. 만약 http로 시작하는 href 값을 가진 앵커 태그를 대상으로 하고 싶다면, 위의 코드와 유사한 선택자를 사용하면 됩니다.

https://로는 찾아지지 않습니다. 이 표현은 부적절하고 https://로 시작하는 url도 마찬가지입니다.

여러분이 사진으로 링크 걸린 앵커 전체에 스타일을 적용하고 싶다면 어떨까요? 그 경우에는 문자열의 끝을 찾아봅시다.

데모 보기

호환성
IE7+

파이어폭스
크롬
사파리
오페라
14. X[href$=".jpg"]
a[href$=".jpg"] {
color: red;
}
문자열 끝에 적용하도록 정규 표현식 기호인 $를 한번 더 사용하겠습니다. 이번 경우에는 이미지(나 최소한 .jpg로 끝나는 url)로 링크가 걸린 앵커 전체를 찾을 것입니다. gif와 png는 영향받지 않습니다.

데모 보기

호환성
IE7+
파이어폭스
크롬
사파리
오페라
15. X[data-*="foo"]
a[data-filetype="image"] {
color: red;
}
8번 내용을 다시 참조합시다. 여러 가지 이미지 형식(png, jpeg, jpg, gif)은 어떻게 적용할 수 있을까요? 다음과 같이 우리는 선택자를 여러 개 만들 수 있습니다.

a[href$=".jpg"],
a[href$=".jpeg"],
a[href$=".png"],
a[href$=".gif"] {
color: red;
}
그런데, 이 방식은 골치 아프고 비효율적입니다. 커스텀 속성을 사용하는 다른 해결 방식이 있습니다. 이미지로 링크 걸린 앵커마다 data-filetype 속성을 넣으면 어떨까요?

 Image Link
그러면 갈고리(hook) 역할을 이용해 해당 앵커만 대상으로 삼는 일반 속성 선택자를 사용할 수 있습니다.

a[data-filetype="image"] {
color: red;
}
데모 보기

호환성
IE7+
파이어폭스
크롬
사파리
오페라
16. X[foo~="bar"]
a[data-info~="external"] {
color: red;
}

a[data-info~="image"] {
border: 1px solid black;
}
친구에게 깊은 인상을 남겨줄 특별한 선택자가 있습니다. 이 요령을 알고 있는 사람은 그리 많지 않습니다. 물결표(~)를 이용하면 띄어쓰기로 구분되는 값이 있는 속성을 대상으로 할 수 있습니다.

15번에 있는 커스텀 속성 방식으로 data-info 속성을 만들면 됩니다. 이 속성은 우리가 메모하는 무엇이든지 띄어쓰기로 구분한 목록을 받을 수 있습니다. 이 경우, 외부 링크와 이미지 링크를 메모할 수 있습니다. 단지 예를 들면 말이죠.

" Click Me, Fool
위의 마크업을 적당한 위치에 쓰면 ~ 속성 선택자 방식을 이용해 두 개의 값 중 하나라도 있는 태그를 대상으로 삼을 수 있습니다.

/* Target data-info attr that contains the value "external" */
a[data-info~="external"] {
color: red;
}

/* And which contain the value "image" */
a[data-info~="image"] {
border: 1px solid black;
}
꽤 훌륭하지요?

데모 보기

호환성
IE7+
파이어폭스
크롬
사파리
오페라

17. X:checked
input[type=radio]:checked {
border: 1px solid black;
}
이 가상 클래스는 라디오 버튼이나 체크박스처럼 체크되는 사용자 인터페이스 요소만을 대상으로 합니다. 아래 코드처럼 간단합니다.

데모 보기

호환성
IE9+
파이어폭스
크롬
사파리
오페라
18. X:after
before과 after 가상 클래스는 매우 효과적입니다. 사람들이 늘 이 두 클래스를 효과적으로 사용하는 새롭고 창의적인 방법을 찾고 있는 듯합니다. 이 클래스는 선택된 요소 주변에 콘텐츠를 생성합니다.

많은 사람이 clear-fix 핵을 접했을 때 이 클래스를 맨 먼저 도입했었습니다.

clearfix:after {

```
content: "";
display: block;
clear: both;
visibility: hidden;
font-size: 0;
height: 0;
}
```

clearfix {

```
   *display: inline-block;
   _height: 1%;
}
이 핵은 요소 뒤에 공간을 덧붙이고 float 효과를 제거하는데 :after 가상 클래스를 사용했습니다. 특히
overflow: hidden; 방법이 불가능한 경우 여러분이 사용할 방법 중에 가장 훌륭한 방법입니다.
```

다른 창의적 방식은 그림자 제작에 관한 간단한 팁을 참조해 보세요.

CSS3 선택자 명세서를 보면, 가상 요소는 엄밀히 말해 두 개의 콜론(::)으로 표현되어야 합니다. 그렇지만, 일관성을 위해 유저 에이전트는 콜론을 하나 사용한 경우도 허용합니다. 사실 현재, 프로젝트에서 콜론이 한 개인 버전을 사용하는 게 더 현명합니다.

호환성
IE8+

파이어폭스
크롬
사파리
오페라
19. X:hover
div:hover {
background: #e3e3e3;
}
에이. 이 선택자는 알고 있겠죠. 공식 용어는 사용자 동작(user action) 가상 클래스랍니다. 혼란스럽겠지만 그렇지는 않습니다. 사용자가 요소 위에 커서를 올릴 때 특정한 스타일을 적용하고 싶나요? 이 선택자로 처리하세요!

알아두세요. 앵커 태그가 아닌 태그에 :hover 가상 클래스를 적용했을 때 인터넷 익스플로러의 하위 버전에서는 반응하지 않습니다.

대부분 hover 상태에서, 가령 앵커 태그에 border-bottom을 적용할 때 이 선택자를 사용합니다.

a:hover {
border-bottom: 1px solid black;
}
꿀팁 - border-bottom: 1px solid black;이 text-decoration: underline;보다 보기 더 좋습니다.

호환성
IE6+ (IE6에서 :hover는 반드시 앵커 요소에 적용해야 합니다.)
파이어폭스
크롬
사파리
오페라
20. X:not(선택자)
div:not(#container) {
color: blue;
}
negation 가상 클래스는 특히 유용합니다. 제가 모든 div를 선택하고 싶은데, 그중에서 id가 container인 것만 빼고 싶다고 합시다. 위의 코드가 그 작업을 완벽하게 수행합니다.

혹은, (권장하지 않지만) 제가 단락 태그만 제외하고 요소 전체를 선택하고 싶다고 한다면 아래처럼 하면 됩니다.

*:not(p) {
color: green;
}
데모 보기

호환성
IE9+

파이어폭스
크롬
사파리
오페라
21. X::가상 요소
p::first-line {
font-weight: bold;
font-size: 1.2em;
}
첫 번째 줄이나 첫 글자와 같이 요소 일부분에 스타일을 적용하는데 가상 요소(::로 표기되는)를 사용할 수 있습니다. 효과를 보려면 이 요소를 반드시 블록 레벨 요소에 적용해야 합니다.

가상 요소는 두 개의 콜론(::)으로 표시됩니다.

단락의 첫 글자
p::first-letter {
float: left;
font-size: 2em;
font-weight: bold;
font-family: cursive;
padding-right: 2px;
}
이 코드는 페이지에 있는 단락을 모두 찾은 다음 해당 요소의 첫 글자만을 대상으로 하는 추상 개념입니다.

신문처럼 글의 첫 글자를 스타일로 꾸미는 데 자주 사용됩니다.

단락의 첫 줄
p::first-line {
font-weight: bold;
font-size: 1.2em;
}
마찬가지로 ::first-line 가상 요소는 요소의 첫 번째 줄에만 스타일을 적용합니다.

기존 스타일 시트와 일관되도록 유저 에이전트는 CSS 레벨 1과 2 (즉 :first-line, :first-letter, :after)에서 도입한 가상 요소의 이전 표기인 하나의 콜론도 수용해야 합니다. 이 명세서에서 도입된 새 가상 요소에는 호환성이 부족합니다. - 출처

데모 보기

호환성
IE6+
파이어폭스
크롬
사파리

오페라
22. X:nth-child(n)
li:nth-child(3) {
color: red;
}
여러 요소 중에서 특정 요소를 지목하는 방법이 없었던 시절이 기억나나요? 그 문제를 풀어줄 nth-child 가상 클래스가 있답니다!

nth-child는 변숫값을 정수(integer)로 받습니다. 0부터 시작하지는 않습니다. 두 번째 항목을 대상으로 하고 싶다면 li:nth-child(2)로 작성합니다.

자식 요소의 변수 집합을 선택하는 데에도 이 방식을 활용할 수 있습니다. 가령, 항목의 4번째마다 선택하려면 li:nth-child(4n)로 작성하면 됩니다.

데모 보기

호환성
IE9+
파이어폭스 3.5+
크롬
사파리
23. X:nth-last-child(n)
li:nth-last-child(2) {
color: red;
}
만약 ul에 항목이 엄청 많고, 여러분은 끝에서 세 번째 항목만 필요하다고 한다면 어떨까요? li:nth-child(397)로 작성하지 말고 nth-last-child 가상 클래스를 쓰면 됩니다.

이 선택자는 16번과 거의 동일합니다. 다만 집합의 끝에서부터 출발하면서 동작한다는 게 다릅니다.

데모 보기

호환성
IE9+
파이어폭스 3.5+
크롬
사파리
오페라
24. X:nth-of-type(n)
ul:nth-of-type(3) {
border: 1px solid black;
}
child를 선택하지 않고 요소의 type을 선택해야 하는 날이 있을 것입니다.

순서를 정하지 않은 목록 5개가 있는 마크업을 상상해 보세요. 세 번째 ul에만 스타일을 지정하고 싶은데 그것을 지정할 유일한 id가 없다면, nth-of-type(n) 가상 클래스를 이용할 수 있습니다. 위의 코드에서 세 번째 ul에만 테두리 선이 둘려집니다.

데모 보기

호환성
IE9+
파이어폭스 3.5+
크롬
사파리
25. X:nth-last-of-type(n)
ul:nth-last-of-type(3) {
border: 1px solid black;
}
일관성을 유지하도록 목록 선택자의 끝부터 출발해 지정한 요소를 대상으로 하는 nth-last-of-type을 사용할 수도 있습니다.

호환성
IE9+
파이어폭스 3.5+
크롬
사파리
오페라

1. X:first-child
   ul li:first-child {
   border-top: none;
   }
   이 구조적 가상 클래스를 이용해 부모 요소의 첫 번째 자식만 대상으로 삼을 수 있습니다. 목록에서 맨 처음과 맨 나중 항목에서 테두리 선을 제거하는데 이 방식을 흔히 사용합니다.

예를 들면, 가로 행 목록이 있다고 합시다. 행마다 border-top과 border-bottom이 적용되어 있습니다. 글쎄요. 그 정렬에서 맨 처음과 마지막 항목이 약간 어색해 보이겠네요.

많은 디자이너가 이를 보완하려고 first와 last 클래스를 적용합니다. 그 대신에 여러분은 이 가상 클래스를 사용하면 됩니다.

데모 보기

호환성
IE7+
파이어폭스
크롬
사파리
오페라

27. X:last-child
ul > li:last-child {
color: green;
}
first-child와 반대로 last-child는 부모 요소의 마지막 항목을 대상으로 합니다.

예제
이 클래스 중에 활용 가능한 사례를 보여주는 간단한 예제를 만들어 봅시다. 스타일이 적용된 항목을 제작하겠습니다.

마크업

 List Item
 List Item
 List Item

그냥 코드입니다. 단순한 목록일 뿐이지요.

CSS
ul {
width: 200px;
background: #292929;
color: white;
list-style: none;
padding-left: 0;
}

li {
padding: 10px;
border-bottom: 1px solid black;
border-top: 1px solid #3c3c3c;
}
이 스타일에 배경을 입히고, 브라우저상에서 ul 기본값을 제거하며, 깊이를 약간 주려고 li마다 테두리 선을 주겠습니다.

Styled List
목록에 깊이를 더하기 위해 각각의 li에 border-bottom을 적용합니다. 이는 그림자가 되거나 li 배경보다 어두운색이 될 것입니다. 다음에 배경보다 더 밝은 값을 border-top에 적용합니다.

단 한 가지 문제점은, 위의 이미지에서 보이듯, 순서에 정해지지 않은 목록의 맨 위와 맨 아래에도 테두리 선이 적용된다는 것입니다. 자연스럽게 보이지 않죠. :first-child와 :last-child 가상 클래스를 사용해 이 문제를 고쳐봅시다.

li:first-child {
border-top: none;
}

li:last-child {
border-bottom: none;
}
Fixed
야아. 고쳐졌군요!

데모 보기

호환성
IE9+
파이어폭스
크롬
사파리
오페라
맞아요. IE8은 :first-child를 지원하지만 :last-child를 지원하지 않습니다. 말도 안 되죠.

1. X:only-child
   div p:only-child {
   color: red;
   }
   솔직히 여러분은 아마 only-child 가상 클래스를 거의 사용하지 않을 것입니다. 그렇더라도 쓸 수 있으니 써봐야 하겠죠.

이 선택자는 부모의 단 하나의 자식 요소를 지정할 수 있습니다. 위의 코드를 참조하면, 가령, div의 단 하나의 자식인 문단만 빨간색으로 칠해질 것입니다.

아래의 마크업을 생각해 봅시다.

 My paragraph here.

 Two paragraphs total.
 Two paragraphs total.

이 경우, 두 번째 div의 문단은 대상이 되지 않고 오직 첫 번째 div가 대상이 됩니다. 하나 이상의 자식을 요소에 적용하는 순간에 only-child 가상 클래스의 효과는 사라지게 됩니다.

데모 보기

호환성
IE9+
파이어폭스
크롬
사파리

오페라

1. X:only-of-type
   li:only-of-type {
   font-weight: bold;
   }
   이 구조상의 가상 클래스는 기발한 방식으로 사용될 수 있습니다. 부모 컨테이너에 형제 요소가 없는 요소를 대상으로 합니다. 예로, 단 하나의 목록 아이템인 ul 전부를 대상으로 삼습니다.

우선, 이 작업을 어떻게 완료할지 자신에게 질문해 보세요. 여러분은 ul li로 하겠지만, 목록 아이템 전체가 대상이 됩니다. 유일한 해결 방법은 only-of-type을 사용하는 것입니다.

ul > li:only-of-type {
font-weight: bold;
}
데모 보기

호환성
IE9+
파이어폭스 3.5+
크롬
사파리
오페라
30. X:first-of-type
first-of-type 가상 클래스로 해당 type의 첫 번째 형제 선택자를 선택할 수 있습니다.

테스트
이해를 돕도록 테스트를 해봅시다. 아래 마크업을 코드 편집기에 복사해 넣으세요.

 My paragraph here.

 List Item 1
 List Item 2


```
   <ul>
      <li> List Item 3 </li>
      <li> List Item 4 </li>
   </ul>
</div>
다음 내용을 읽기 전에 "List Item 2"만 대상으로 하는 방법을 생각해 보세요. 생각났다면 (혹은
포기했더라도) 다음으로 넘어갑니다.
```

해결 방법 1
이 테스트를 푸는 방법은 여러 가지입니다. 이 중에서 몇 가지를 살펴보겠습니다. first-of-type을 사용해서 시작해 보지요.

```
ul:first-of-type > li:nth-child(2) {
font-weight: bold;
}
```

이 코드는 기본적으로 "페이지에서 순서를 중요시하지 않는 첫 번째 목록을 찾고 나서 목록 아이템인 직계 자식만 찾아라."라고 이야기합니다. 그다음, 그 결과 세트에서 두 번째 목록 아이템만 걸러냅니다.

해결 방법 2
다른 방법은 인접 선택자를 사용하는 것입니다.

```
p + ul li:last-child {
font-weight: bold;
}
```

이 시나리오에서는 p 태그 바로 뒤에 있는 ul을 찾고 나서 그 요소의 가장 마지막 자식을 찾습니다.

해결 방법 3
이 선택자를 써서 원하는 대로 불쾌해하거나 쾌활해 할 수 있습니다.

```
ul:first-of-type li:nth-last-child(1) {
font-weight: bold;
}
```

이번에는 페이지에 있는 첫 번째 ul을 잡고 나서 가장 첫 번째 목록 아이템을 찾습니다. 바로 아래부터 시작해서요! :)

데모 보기

B.2. CSS Unit (반응형)

CSS Unit (CSS 7가지 단위)

우리가 잘 알고 있는 CSS기술을 사용하는 것은 쉽고 간단할 수 있지만 새로운 문제에 봉착하게 되면 해결하기 어려울 수 있습니다.

웹은 항상 성장,변화하고 있고 새로운 해결방안 역시 계속 성장하고 있습니다.

그렇기 때문에 웹 디자이너와 프론트 엔드 개발자가 습득한 노하우를 활용할 수 밖에 없다는 것을 잘 알고 있습니다.

특별한 방법을 알면서, 단 한 번도 사용하지 않더라도 언젠가 필요한 때가 오면 정확한 방법을 실무에 적용할 수 있다는 뜻이기도 합니다.

이 글에서는 이전엔 알지 못했던 몇 가지의 CSS 방법에 대해 알아보고자 합니다.

몇몇 수치 단위들은 픽셀이나 em과 비슷하지만 다른 방법에 대해 살펴보도록 합니다.

rem(root em)

여러분에게 조금 익숙할 수 있는 단위로 시작해 보자면 em은 현재의 font-size를 정의합니다.

일례로, body 태그에 em값을 이용해 폰트 사이즈를 지정하면 모든 자식 요소들은 body의 폰트 사이즈에 영향을 받습니다.

HTML
<body>
Test
</body><
CSS
body {
font-size: 14px;
}
div {
font-size: 1.2em; // calculated at 14px * 1.2, or 16.8px
}
여기, div에 font-size를 1.2em으로 지정했습니다. 이 예제에서는 14px을 기준으로 1.2배의 폰트 사이즈로 표현이 됩니다.

결과적으로 16.8px의 크기가 됩니다.

그런데 여기 em으로 정의한 폰트 사이즈를 각각의 자식에 선언하면 어떤 일이 생길까요?

같은 CSS를 적용한 동일한 코드를 추가해보았습니다.

각각의 div는 각 부모의 폰트 사이즈를 상속받아 점점 커지게 됩니다.

HTML

Test (14 * 1.2 = 16.8px)

Test (16.8 * 1.2 = 20.16px)

Test (20.16 * 1.2 = 24.192px)

이것은 어떤 경우엔 바람직하겠지만 대부분의 경우, 단순하게 단일 사이즈로 표현하기도 합니다.

이런 경우 바로 rem 단위를 사용하면 됩니다.

rem의 "r"은 바로 "root(최상위)"를 뜻합니다.

최상위 태그(요소)에 지정한 것을 기준으로 삼으며, 보통 최상위 태그는 html태그입니다.

CSS
```
html {
font-size: 14px;
}
div {
font-size: 1.2rem;
}
```
이전 예제에서 만든 복잡한 단계의 세 div는 모두 16.8px의 폰트 사이즈로 표현될 것입니다.

이 rem unit은 그리드 시스템에서도 유용하게 사용가능합니다.

rem은 폰트에서만 사용하진 않습니다.

예를 들어, 그리드 시스템이나 rem을 이용한 기본 폰트 사이즈 기반으로 만든 UI 스타일, 그리고 em을 이용해 특정 위치에 특별한 사이즈를 지정할 수도 있습니다.

보다 정확한 폰트 사이즈나 크기 조정을 가능하게 해 줄 것입니다.

CSS
```
.container {
width: 70rem; /* 70 * 14px = 980px */
}
```
개념적으로 보면, 이 아이디어는 여러분의 콘텐츠 사이즈를 조절 할 수 있는 인터페이스 전략과 유사합니다. 그러나 모든 경우에 반드시 이런 방법을 따를 필요는 없습니다.

rem (root em) 단위의 호환성은 caniuse.com에서 확인할 수 있습니다.

vh & vw (vertical height & vertical width)

반응형 웹디자인 테크닉은 퍼센트 값에 상당히 의존하고 있습니다.

하지만 CSS의 퍼센트 값이 모든 문제를 해결할 좋은 방법은 아닙니다. CSS의 너비 값은 가장 가까운 부모 요소에 상대적인 영향을 받습니다.

만약 타켓 요소의 너비값과 높이값을 뷰포트의 너비값과 높이값에 맞게 사용할 수 있다면 어떨까요?

바로 vh와 vw 단위가 그런 의도에 맞는 단위이고 vh 요소는 높이값의 100분의 1의 단위입니다.

예를 들어 브라우저 높이값이 900px일때 1vh는 9px이라는 뜻이 되지요. 그와 유사하게 뷰포트의 너비값이 750px이면 1vw는 7.5px이 됩니다.

이 규칙에는 무궁무진한 사용방법이 있습니다.

예를 들면, 최대 높이값이나 그의 유사한 높이값의 슬라이드를 제작할때 아주 간단한 CSS만 입력하면 됩니다.

CSS
.slide {
height: 100vh;
}

스크린의 너비값에 꽉 차는 헤드라인을 만든다고 가정해 봅니다.

vw로 폰트 사이즈를 지정하면 쉽게 달성할 수 있습니다.

해당 사이즈는 브라우저의 너비에 맞춰 변할 것입니다. (브라우저 크기를 늘였다 줄였다 해보세요)

뷰포트 vw, vh 단위의 호환성은 caniuse.com에서 확인할 수 있습니다.

vmin & vmax

vh와 vw이 늘 뷰포트의 너비값과 높이값에 상대적인 영향을 받는다면 vmin과 vmax는 너비값과 높이값에 따라 최대, 최소값을 지정할 수 있습니다.

예를 들면 브라우저의 크기가 1100px 너비, 그리고 700px 높이일때 1vmin은 7px이 되고 1vmax는 11px이 됩니다.

너비값이 다시 800px이 되고 높이값이 1080px이 되면 vmin은 8px이 되고 vmax는 10.8px이 됩니다.

어때요, 이 값들을 사용할 수 있나요?

언제나 스크린에 보여지는 요소를 만든다고 가정해 봅니다.

높이값과 너비값을 vmin을 사용해 100으로 지정합니다.

예를 들어 터치화면 양 변에 가득차는 정사각형 요소를 만들때는 이렇게 정의하면 됩니다.

CSS
.box {
height: 100vmin;
width: 100vmin;
}

만약 커버처럼 뷰포트 화면에 보여야 하는(모든 네 변이 스크린에 꽉 차 있는) 경우에는 같은 값을 vmax로 적용하면 됩니다.

CSS
.box {
height: 100vmax;
width: 100vmax;
}

알려드린 이 규칙들을 잘 조합해 활용하면 뷰포트에 맞는 매우 유연한 방식으로 사이즈 조절을 가능하게 할 수 있습니다.

뷰포트 단위: vmin, vmax의 호환성은 caniuse.com에서 확인할 수 있습니다.

vh : 1/100th of the height of the viewport.

vw : 1/100th of the width of the viewport.

vmin : 1/100th of the minimum value between the height and the width of the viewport.

vmax : 1/100th of the maximum value between the height and the width of the viewport.

Remind : vw, vh, vmin, vmax

뷰포트(Viewport)를 기준으로 하는 길이(length) 값으로, 문서 또는 모바일 기기 에서 볼 수 있는 부분의 크기를 기준으로 크기를 설정합니다.

각 속성의 풀네임은 다음과 같습니다.

VW(Viewport Width) : 뷰포트 너비의 1% 길이와 동일합니다.
VH(Viewport Height) : 뷰포트 높이의 1% 길이와 동일합니다.
VMIN(Viewport Minimum) : 뷰포트 너비 또는 높이를 기준으로 하는 최소 값입니다.
VMAX(Viewport Maximum) : 뷰포트 너비 또는 높이를 기준으로 하는 최대 값입니다.
vmin, vmax 값은 뷰포트의 너비, 높이 길이 중 '작은' 혹은 '큰' 길이를 기준으로 값이 동적으로 설정됩니다.

예를 들어 1000 × 500 크기의 뷰포트가 있다고 했을 때 1vmin 값은 5px 이고, 1vmax 값은 10px로 계산됩니다.

크기가 1200 × 570 으로 변경되면 1vmin은 5.7px, 1vmax는 12px이 됩니다.

즉, 화면 크기에 상대적으로 변경되는 단위가 뷰포트 단위입니다.

참고로 IE 9-11 에서 vmax는 제대로 지원하지 못합니다.

Edge 브라우저에서 온전하게 vmax를 지원합니다.

IE 5.5 이상 제대로 뷰포트 단위를 지원하게 하려면 폴리필 vminpoly를 사용할 수 있습니다.

ex & ch

ex와 ch 단위는 현태 폰트와 폰트 사이즈에 의존한다는 점에서 em 그리고 rem과 비슷합니다.

em과 rem과 다른 점은 이 두 단위가 font-family에 의존한다면 다른 두 단위는 폰트의 특정 수치에 기반한다는 점입니다.

ch 단위, 또는 글꼴 단위는 제로 문자인 0의 너비값의 "고급 척도"로 정의됩니다.

흥미로운 의견은 에릭 마이어의 블로그에서 확인할 수 있습니다. 그러나 기본 컨셉은 monospace 폰트의 N 의 너비값을 부여 받았다는 것이며, width: 40ch;는 40개의 문자열을 포함하고 있다는 뜻입니다.

이 특정 규칙은 점자 레이아웃에 기반하고 있지만, 이 기술의 가능성은 간단한 어플리케이션 그 이상으로 확장할 수 있습니다.

ex 단위의 정의는 "현재 폰트의 x-높이값 또는 em의 절반 값"이라고 할 수 있습니다. x-높이값은 소문자 x의 높이값이기도 합니다.

폰트의 중간 지점을 알아내기 위해 자주 사용하는 방법입니다.

x-높이값; 소문자 x의 높이값 (자세한 것은 웹 타이포그래피의 해부학 링크를 참조하세요)

이 단위는 타이포그래픽에서 세밀한 조정을 할 때 많이 사용합니다.

예를 들어, 위첨자 태그인 sup 에게 position을 relative로 하고 bottom 값을 1ex라고 하면 위로 올릴 수 있습니다.

아래첨자 역시 비슷한 방법으로 아래로 내릴 수 있습니다.

브라우저는 위첨자와 아래첨자의 기본값을 vertical-align으로 정의하고 있지만, 보다 정교한 사용법을 알고 싶다면 아래와 같이 작성할 수 있습니다.

CSS
```
sup {
position: relative;
bottom: 1ex;
}
sub {
position: relative;
bottom: -1ex;
}
```

사용 가능 여부

ex는 CSS1부터 있던 단위였고, ch 단위는 아직 찾을 수 없습니다.

에릭 마이어의 블로그에 있는 CSS 단위와 값에서 보다 많은 상세 정보를 볼 수 있습니다.

마치며

여러분의 막강한 CSS 도구들의 무한한 확장과 지속되는 개발환경에 지속적으로 살펴보시기 바랍니다.

아마 특정 문제를 해결하기 위해 예상치 못한 해결방법으로 이 애매한 특정 단위들을 사용할 수도 있을 것입니다.

새로운 스펙들에 대해 시간을 투자해 보시기 바랍니다.

그리고 cssweekly와 같은 좋은 사이트에도 가입해서 지속적인 뉴스를 업데이트 받아보시기 추천합니다. 그리고 주간 업데이트에 가입하는 거 잊지 마세요.

무료 튜토리얼과 Tuts+의 웹디자인에서 나오는 다양한 자료들을 만날 수 있습니다.

NOTE 출처: https://webclub.tistory.com/356 [Web Club]

B.3. CSS Targeting Example

```
p {
  color: var(--subtitle-color, blue);
} /* <p> [CSS] */
```

```
#iA {
  color: var(--subtitle-color, blue);
} /* <element id="iA"> [CSS] */
```

```
[href] {
  color: var(--link-color, blue);
} /* <element href> [CSS] */
```

```
.subtitle {
  color: var(--subtitle-color, blue);
} /* <element class="subtitle"> */
```

```
a#iA ul.subtitle a[style='page-break-after: always'] {
  color: var(--subtitle-color, blue);
} /* <a id="iA"> ... <ul class="subtitle"> ... <a style = 'page-break-after: always'[CSS]>
*/
```

B.4. CSS Calculation Example

```
.banner {
  width: calc(100% - 80px);
  --widthA: 100px;
  --widthB: calc(var(--widthA) / 2); /*50px*/
  --widthC: calc(var(--widthB) / 2); /*25px*/
}
```

B.5. min()

B.6. max()

B.7. :is() / :where()

B.8. ::before / ::after

B.9. 추가

출처: http://blog.hivelab.co.kr/%EA%B3%B5%EC%9C%A0before%EC%99%80after-%EA%B7%B8%EB%93%A4%EC%9D%98-%EC%A0%95%EC%B2%B4%EB%8A%94/
① :first-child(가상클래스) – class를 지정하지 않아도 li의 첫번째 자식요소를 선택하여 제어할 수 있습니다.
② ::first-letter(가상요소) – li내의 첫번째 글자를 감싸고 있는 요소가 없어도 있는 것과 같이 제어해줄 수 있습니다.
① ::before – 실제 내용 바로 앞에서 생성되는 자식요소
② ::after – 실제 내용 바로 뒤에서 생성되는 자식요소

::before와 ::after를 쓸 땐 content라는 속성이 꼭 필요하다고 합니다!
content는 또 어떤 것인지 알아보실까요?

1.3) content="" 란?

::before와 ::after와 꼭 함께 쓰이는 'content'는 '가짜' 속성입니다.
HTML 문서에 정보로 포함되지 않은 요소를 CSS에서 새롭게 생성시켜주기 때문이죠!

아래의 표는 content를 쓸 때, 대표적으로 사용되는 속성들입니다.
2.1) gnb 구분 bar 넣기

NOTE

구분 bar가 포함된 서브네비게이션(snb, BreadCrumb)등 을 구성할 때 after를 사용한다면, 따로 클래스를 선언하지 않아도 쉽게 구현할 수 있습니다.
3
– 사용법 : li에 after와 content를 사용하여 바(|)를 선언 후, last-child를 이용하여 마지막 li의 content를 재선언 해줍니다.
– 이슈 : last-child는 IE9부터 지원합니다.
– 이슈해결 : before와 IE7, 8까지 지원되는 first-child를 활용법으로 변경할 수 있습니다.

아래와 같이 이슈해결이 가능합니다.
3-1

2.2) 앞,뒤에 추가 정보를 넣는 방법

특정 컨텐츠 앞, 뒤에 붙여지는 추가 정보들을 넣을 때도 편리하게 쓸 수 있습니다.
4
– 사용법 : 요소의 앞/뒤에 before 혹은 after를 선언합니다.. content=""에 넣고자 하는 문구를 입력해줍니다.
– 이슈 : 강조하고 싶은 중요한 정보가 담겼다면, content를 스크린리더기에서도 꼭 읽어주어야 할텐데, 과연 스크린리더기에서 content의 내용을 읽어줄까요?
– 이슈해결 : 목차 3번, 접근성이슈에서 관련 내용을 읽어보시고, 답을 찾아 보실 수 있습니다! 투비컨티뉴!

Chapter 8. SASS (SCSS)

CSS 전처리기

8.1. 명령어

- 변환
  sass custom.scss custom.css
- 실시간 변환 (감시)
  sass --watch custom.scss custom.css
- .map
  소스맵(Sourcemap) 파일입니다. 컴파일된 소스를 원본 소스로 맵핑해 주는 역할, 원래 소스가 어디에 있는지 보여주는 지도
  - .map 없애기
    sass --no-source-map custom.scss custom.css

> **NOTE** 출처: https://nykim.work/97

8.2. 문법

8.2.1. Variables (변수)

$ 기호를 사용하여 스타일시트에서 재사용하려는 정보를 저장.

```
$font-stack: Helvetica, sans-serif;
$primary-color: #333;
body {
  font: 100% $font-stack;
  color: $primary-color;
}
```

```
body {
  font: 100% Helvetica, sans-serif;
  color: #333;
}
```

8.2.2. Nesting (중첩)

HTML과 동일한 시각적 계층 구조를 따르는 방식으로 CSS 선택기를 중첩

```
nav {
  ul {
    margin: 0;
    padding: 0;
    list-style: none;
  }

  li { display: inline-block; }

  a {
    display: block;
    padding: 6px 12px;
    text-decoration: none;
  }
}
```

```
nav ul {
  margin: 0;
  padding: 0;
  list-style: none;
}
nav li {
  display: inline-block;
}
nav a {
  display: block;
  padding: 6px 12px;
  text-decoration: none;
}
```