**meta태그**

charset : 문자 인코딩 방식 표현

name : content에 들어가는 내용 이름

content : 텍스트 기입

h1~h6 heading

p paragraph

* 연속 띄어쓰기나 줄바꿈은 entity 기호 써야함

**entity**

&amp; : & (ampersand) : 컴퓨터가 &기호를 만나면 엔티티표현을 준비하기 떄문

&lt; : < less than : 부등호 기호는 태그 시작에 쓰임

&gt; : > greater than

&quot; : “ quotation mark : 따옴표는 속성 표현에 쓰임

&apos; : ‘ apostrophe

&nbsp; : no-break-space : 여러 개의 띄어쓰기를 HTML은 하나로 인식함

&copy; : 저작권기호 … 이런 특수기호가 많이 있다~

<**img** src=”이미지주소” alt=”대신 보여줄 텍스트 : 웹 접근성 필수요소” />

<a href=” 하이퍼링크주소” target=”새 창 or 현재 창” download=””>

감쌀 요소들(여러 개 가능) </a>

ul (unordered list) 밑에 li (list item), ol (ordered list) 밑에 li (list item)

table > thead > tr > th , td

> tbody > tr > th, td

**CSS 박스모델**

width, height 으로 조절되는 크기는 : border + padding + content 다 더한거. margin은 따로.

padding과 margin을 늘리면 content 영역이 줄어듦. 0까지 줄어들수있음. 내용은 걍 삐져나옴

content : 실제 내용이 들어가는 영역

padding : border와 content 사이

margin : border와 다른 요소 사이의 여백

* margin collapse : 상하로 인접한 두 블록요소의 여백이 겹치면 따로 반영되지 않고 겹쳐서 반영된다

border 안쪽으로 background 정의

padding, margin 순서 : left, right, top, bottom

1개 : 상하좌우 / 2개 : 상하 좌우 / 3개 : 상 좌우 하 / 4개 : 상 우 하 좌 (시계방향)

border : width, style, color 순서대로 모두 지정

border-style : solid / dotted / dashed

**CSS 길이 단위**

px : 10px이 최소크기

em : 바로 위 상위요소 기준 몇 배인지

rem : 최상위요소(html태그) 기준 몇 배인지

vw : 화면 전체 너비를 백등분

vh : 화면 전체 높이를 백등분

vmin : 너비, 높이 중 짧은 쪽 백등분

vmax : ‘’ 긴 쪽 백등분

block : 한 줄 전체를 차지. 새로운 줄에 배치됨. p, h1~h6, div 등

inline : 한 줄 안에서 줄바꿈 없이 배치. 물론 너비가 모자라면 밑으로 내려감. 하위에 block 요소 넣으면 안됨. span, img 등

font-familly : 폰트1, 폰트2, 폰트3 ;

font-size, font-weight, font-style, color…

font-color은 없음 색상은 color !!!

color : #000000 ~ #FFFFFF 16진수 red, green, brue, alpha(투명도)

3자리 : rgb / 4자리 : rgba / 6자리 : rrggbb / 8자리 : rrggbbaa

16x16=256 이니까 0~255로 표현 가능. => 함수로 지정 가능 rgba( 0~255, 0~255, 0~255, 0~1 )

컨텐츠, border, background에 색상 지정 가능. background는 border 기준으로 안쪽 색칠

display : inline-block : 요소를 인라인 박스로 표시하지만 블록 박스처럼 너비와 높이 지정 가능

display : flex 요소를 유연한 박스로 표시하고 자식 요소들의 크기와 순서를 조절 가능

FOUT / FOIT 나옴. 각각 시간대에 따라서 어떻게 그려지는지 그리고 설명할 수 있어야함

\*\*\*\*\*\*\*\* 브라우저가 켜진 시점, 내용이 로딩된 시점, font 파일이 렌더링 된 시점 3가지 상황에서 각각 옵션별로 어떻게 화면에 그려지고 있는지

예를들어 이 시점에서는 맑음고딕(기본)으로 보이다가 언제 나눔고딕(적용)으로 바뀌는지 상황 설명. 웹 폰트가 다운로드가 됐는지 안됐는지

몇 초간 안보이게 돼있는지,

접속 시점 ... 폰트 다룬로드 시점... block 설정시간... 웹폰트 로딩에 걸린 시간... 등등 시간대에 따라 어떻게 보이는지 작성 \*\*\*\*\*\*\*\*

FOUT : Flash Of **Unstyled** Text

처음에 기본 폰트로 로딩, 웹폰트 다운로드 완료 후 다시 렌더링하여 적용 -> 화면이 깜빡거림

FOIT : Flash Of **Invisible** Text

웹폰트 다운로드 전까지는 글자를 보여주지 않다가, 다운 완료 후 한 번에 보여줌

FOUT 대응 위한 font-display

auto : IE – FOUT , Chrome – FOIT

block : 폰트가 다운로드 될 때까지 숨김. 다운되면 보여줌 (폰트가 중요한 경우)

swap : 우선 기본폰트로 렌더링, 다운완료되면 적용

fallback : 짧은시간 안보이게, 일정시간 안에 다운되면 적용. 시간 안에 실패하면 기본폰트 유지

optional : fallback과 유사하게 동작하지만, 네트워크 상황에 따라 브라우저가 폰트 다운로드 여부 결정. 기본 폰트를 유지할지 웹폰트를 적용할지

**CSS Selector**

#id

.class

A : 모든 A태그

A, B : 모든 A태그랑 B태그 여러 개 동시 선택 가능

A B : 깊이 상관없이 A 하위에 있는 모든 B 태그

A > B : A 바로 아래 단계에 있는 B 태그

\* : 전체

A + B : A의 형제 태그 중 바로 다음에 오는 B 태그

A ~ B : A의 형제 태그 중 다음에 오는 모든 B 태그

first / nth / last / only -child / -last-child / -of-type

n 자리엔 even, odd, an+b 꼴 다 됨.

:first-child : 모든 첫째

plate :first-child (띄어쓰기 필수) 플레이트 자식 중에 첫째

orange:first-child 첫째인 오렌지

:only-child : 외동

A :only-child : A의 자식 중 외동 (띄어쓰기 때문)

:nth-child(n) : 모든 n번째

:nth-last-child(n) : 모든 뒤에서 n번쨰

A:nth-last-child(3) : A태그 중에 3번째 (X) . ( 이건 A:nth-of-type(3) )

뒤에서 3번째인 A

A:nth-of-type(n) : A 중에 n번째

A:only-of-type : A 중에 외동

A:last-of-type : A 중에 막내

:empty

:not(X) : X 자리에 제외할 selector

[attribute]

A[attribute]

[attribute=”value”]

[attribute^=”value”] : attribute 속성의 값(value)이 “value”로 시작하는 태그

[attribute$=”value”] : “” “value”로 끝나는 태그

[attribute\*=”value”] : “” “value”를 포함하는 태그

[attribute~=”value”] : “” “value”를 단어로 포함하는 태그

하나의 태그에 여러 개의 클래스 적용할 수 있다. (CSS우선순위 적용)

class=”class1 class2 class3” 이렇게 지정

.class1.class2.class3 : 세 클래스 다 가지고 있는 태그만 선택. 띄어쓰기 들어가면 달라짐!!

**CSS 우선순위**

!important >> Inline Style > ID > class, attribute, pseudo-class > 태그, ::before, ::placeholder

!important : 스타일 마지막에 붙여 사용. important끼리 겹치면 css우선순위 따름

우선순위가 같은 경우 마지막에 지정한 속성이 적용됨

**position** \* https://www.daleseo.com/css-position/

속성별로 배치 기준에 따라 top/left/bottom/right 로 위치 지정

static : 기본. inline 좌->우, block 상->하. 위치를 임의로 지정할 수 없고 다른 태그와의 관계에 따라 자동으로 배치

relative : 기본 문서 흐름대로 배치한 후, 원래 있던 위치를 기준으로 좌표 지정

absolute : 절대 좌표로 위치 지정. 배치 기준 : position 속성이 static이 아닌 첫 번째 상위 요소

따라서 어떤 요소의 position 속성을 absolute로 지정하면 부모 요소의 position 속성을 relative로 지정해주는 것이 관례.

fixed : 스크롤과 상관없이 좌표 고정. 배치기준 : 뷰포트(viewport), 즉 브라우저 전체화면

sticky : 스크롤 시 경계에 도달하면 고정됨

position 속성이 absolute 또는 fixed로 지정된 요소는 더 이상 앞 뒤의 요소와 상호작용 하지 않는다.

display: none 렌더링하지 않는다 = 화면에 안보여준다

**flex** \* <https://studiomeal.com/archives/197>

flex-direction : row/row-reverse/col/col-reverse

flex-wrap : nowrap main-axis 방향으로 벗어남 (박스 밖으로 삐져나감)

flex-wrap : wrap cross-axis 방향으로 벗어남 (밑으로 내림)

flex-wrap : wrap-reverse cross-axis 반대방향으로 벗어남 (위로 올림)

justify-content

align-items

브라우저 렌더링

Parsing : HTML -> DOM트리, CSS -> CSSOM트리 구축

* Tokenizing
* Lexing
* DOM 생성

Style : 두 트리 결합하여 Render 트리 구축

실제로 화면에 그려지지 않는 요소는 제외

Layout : CPU를 이용해 각 노드의 위치와 크기 계산, Render tree에 반영

Paint : 계산된 값으로 각 노드를 화면상의 실제 픽셀로 변환

변환된 결과는 여러 개의 layer로 관리됨

Composite : layer 합성하여 실제 화면에 나타냄

모든 layout, paint 정보를 활용해서 모니터 화면에 픽셀단위에 표현하는 작업

모니터 화면이 옮겨지면, 경우에 따라 추가로 그림.

경우에 따라 GPU를 활용하여 최적화 함

Reflow : 렌더링 이후 사용자의 다양한 액션에 의해 발생되는 이벤트로 인해 Render tree나 Layout tree 정보가 변경될 때 렌더링 트리 생성과 레이아웃 과정을 다시 수행

* viewport 변경 (브라우저 크기 변경)
* 폰트 크기 변경
* 스크롤
* 화면 확대 / 축소
* 스크립트에 의한 크기 정보 변경
* 새로운 DOM 노드 추가

Repaint : 이미 계산한 layout 영역과 Paint Record 안에서 화면에 그릴 요소들을 일부분 수정

* transform, filter 등의 요소를 변경/추가할 때 발생

reflow는 단지 변경사항을 반영하기 위해 렌더링 트리를 재생성하고 레이아웃 과정을 다시 수행하는 것이고, 실제 이 결과가 화면에 그려지기 위해서는 다시 페인팅 단계를 수행해야 하는데, 이게 repaint. reflow가 일어나면 반드시 repaint가 일어나야 한다.

레이아웃에 영향을 미치지 않는 단순한 색상변경과 같은 변경사항은 reflow 없이 바로 repaint만 일어난다

Reflow 유발하는 속성들 : position, width, height, margin, padding, border, border-width, font-size, font-weight, line-height, text-align, overflow 등

Repaint만 일어나는 속성들 : background, color, border-style 등

콘솔 창 실습

$0 : 브라우저 inspector로 선택한 요소

confirm(“”) 확인->true, 취소->false 반환

append() 바로 밑에(형제) 텍스트도 추가가능

appendChild() 자식 노드에 노드만 추가 가능

innerHTML : 태그 내부 HTML

outerHTML : 태그 본인과 어트리뷰트까지 포함 + 내부 HTML

<input type=”radio” id=”” name=”그룹명” value=””> 같은 그룹 안에서 택1. 그룹명 다르면 따로 선택이 가능해져버림

<label for=”묶일놈id”>텍스트</label>

console.time 타이머 시작

console.timeEnd 타이머 종료 및 출력

script 태그의 위치에 따른 실행 차이

head 안에 있는 스크립트 코드는 웹사이트가 로딩되기 전에 우선적으로 실행되기 때문에 document elemet를 찾을 수 없는 경우가 발생한다

body 안에 있는 스크립트 코드는 위치에 따라 순서대로 실행되기 때문에

body 마지막에 스크립트 태그를 두면 내부의 html이 전부 다 실행 된 다음에 script가 실행된다는 것이 보장된다.

예를들어 body 안에서 선언한 변수명을 사용하는 스크립트코드는 head에 있으면 안되고, body 안에서 변수가 선언된 다음에 위치해야함. (그래서 보통 body 마지막에)

자바스크립트가 아래쪽에서 길게 늘어지면 가독성이 떨어지기 때문에

head에 위치하면서 문제 해결방법 : window.onlad = function() { 여기다 넣기 }

위와 같이 window.onload 함수를 오버라이딩 해주면 해당 함수 내의 코드 스크립트는 웹브라우저 내의 모든 요소가 준비가 되어야 실행이 된다

라이브러리 차트 활용

for문으로 데이터 채우기

const labels = [];

const data = [];

const labels2 = [];

const data2 = [];

for (var i = 0; i<25; i++ ){

    label.push(res2.RealtimeCityAir.row[i].MSRSTE\_NM);

    data.push(res2.RealtimeCityAir.row[i].PM10);

    label2.push(res2.RealtimeCityAir.row[i].MSRSTE\_NM);

    data2.push(res2.RealtimeCityAir.row[i].PM25);

}

const ctx = document.getElementById("myChart");

new Chart(ctx, {

        type: "bar",

        data: {

          labels: labels,

          datasets: [

            {

              label: 'PM10',

              data: data,

              borderWidth: 1,

            },

          ],

        },

자바스크립트 동작원리

,,,? task 큐 제외하면 뭐가있지

Animation Frame : 사용자가 네트워크 요청으로 처리하지 않는 repaint 동작들을 처리하는 큐

비동기 처리