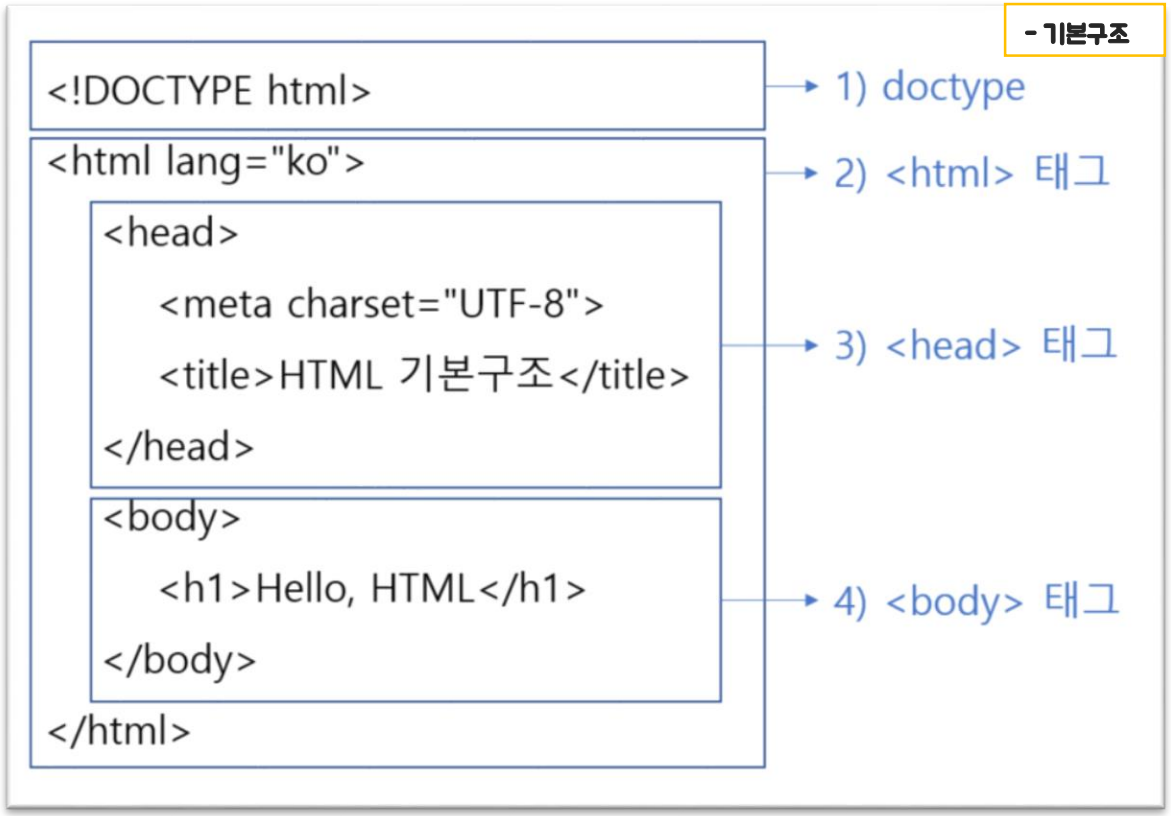
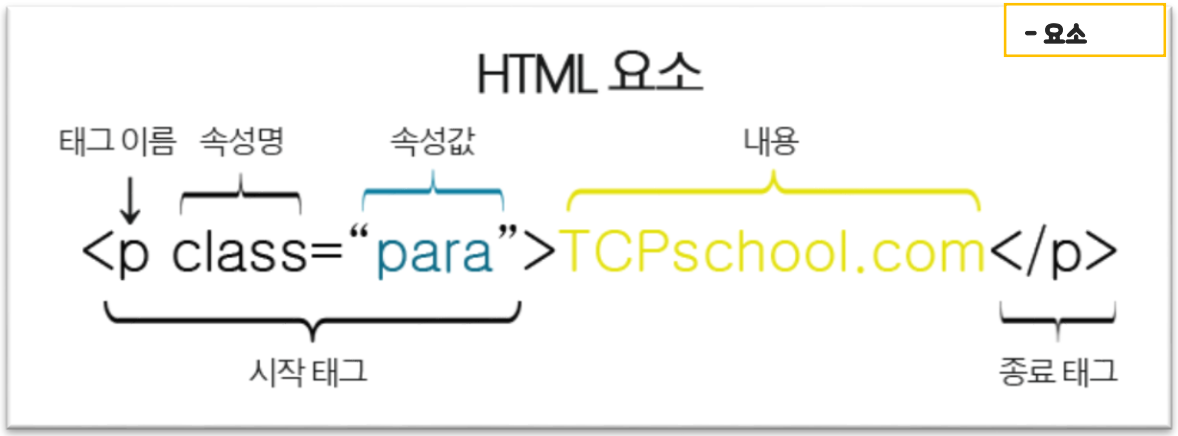
HTML

1.정의

- 기본구조



- 요소



1.정의

-HTML : [Hypertext Markup Language]
Hypertext : 하이퍼링크가 가능 텍스트
Markup : 화면 출력하기 위한 구조
Language : 언어

2.주석

-주석 : 컴파일[실행] 과정에 포함 안되는 코드
1. 주석 형태 : <!-- -->
2.단축키 : 범위 + ctrl+shifc + /

3.기본구조

```
<!DOCTYPE html> <!-- HTML5 버전 명시 -->
<html> <!-- html 코드 전체 구역 [ 반드시 필요 ] -->
    <head> <!-- 페이지제목, 설명 , 외부파일 , 링크 등 표시되는 구역 [ 설정값 ] -->
    <meta charset= "EUC-KR"> <!-- 한글 인코딩[번역기] EUC-KR vs UTF-8 -->
    <title>상단 페이지 제목</title> <!--페이지 제목 -->
</head> <!-- head tag end -->
<body> <!-- web page content : 웹페이지 내용 -->
    <h1> 글 제목1 태그 입니다. </h1>
</body> <!-- body tag end -->
</html> <!-- html tag end -->
```

4.마크업 사용법

< > : 화살표 괄호 사용 [시작 과 끝 존재]
1. <tagname> : 태그 시작 </tagname> : 태그 끝
2. <tagname /> : 태그 시작끝 동일한경우

2.마크업 종류

1. 상대[현재파일기준] 경로

./ : 현재폴더 [생략 가능]

../ : 상위폴더

예)

public

html

현재HTML

목적지파일1

img

목적지파일3

목적지파일2

* 현재HTML 기준에서 목적지파일들의 경로 예

1. ./목적지파일1
2. ../목적지파일2
3. ../img/목적지파일3

2. 절대 경로

C:₩Users₩504t₩git₩ezen_web_2022_B₩frontend₩public₩img₩강아지1.jpg

1.<h1> ~ <h6>

제목 태그 [기본값 : 글꼴크기 , 진하게 , 여백 등] <h1> ~ <h6>

2.<p>

문단/단락 [p : paragraph]

3.

줄바꿈

4.<hr/>

구분선

5.

이미지

6.<audio >

오디오

7.<video >

비디오

8.<source src='영상/비디오경로' />

첨부파일 경로

9.<iframe />

외부 멀티미디어

10. 텍스트

하이퍼링크

1.

순서번호 없는 리스트

2.

순서번호 있는 리스트

3.

리스트의 항목(값)

4.<form>

폼/양식 형식

5.<input />

입력상자

6.<select >

선택상자

7.<option >

선택상자 항목(값)

8.< textarea >

긴글텍스트

9.<fieldset >

양식 그룹

10.<legend >

양식 그룹 타이틀

- 타입 종류

태그명	결과	설명
<form></form>	안 보임	폼 양식의 시작과 끝
<input type=text>	happylucy	한 줄 입력 상자
<input type=password>	****	*로 표시되는 암호 입력상자
<input type=raido>	회사	라디오 버튼
<input type=checkbox>	영화감상	체크상자
<input type=button>	버튼	일반 버튼
<input type=submit>	쿼리 전송	전송확인 버튼
<input type=reset>	원래대로	초기화 버튼
<input type=image>	중복확인	이미지 버튼
<input type=file>	찾아보기...	파일 업로드
<input type=hidden>	안 보임	숨겨진 필드
<textarea></textarea>		여러 줄 입력상자
<select></select>	2002	목록 상자의 시작과 끝
<option>	2002	목록 상자에 포함될 항목 지정

- 속성 종류

속성 (Attribute)	기능
disabled	선택이나 수정이 불가능한 상태로 지정합니다.
max min	number 타입에서 최대값/최소값을 지정 합니다. setp속성과 병행 가능
maxlength	입력값의 글자 수 최대 길이를 지정합니다
readonly	키보드로 수정될수 없게 속성을 부여합니다. (<script>에서 수정가능)
required	반드시 입력되어야 form 전송을 가능하게 합니다.
size	특정 길이의 값을 입력하도록 설정 가능합니다
step	number 타입에 간격(interval) 지정 가능합니다
title	마우스를 올렸을 때 나타날 메세지를 지정합니다
value	전송할 입력값을 지정합니다

1.<table>

테이블

2.<tr>

테이블 행

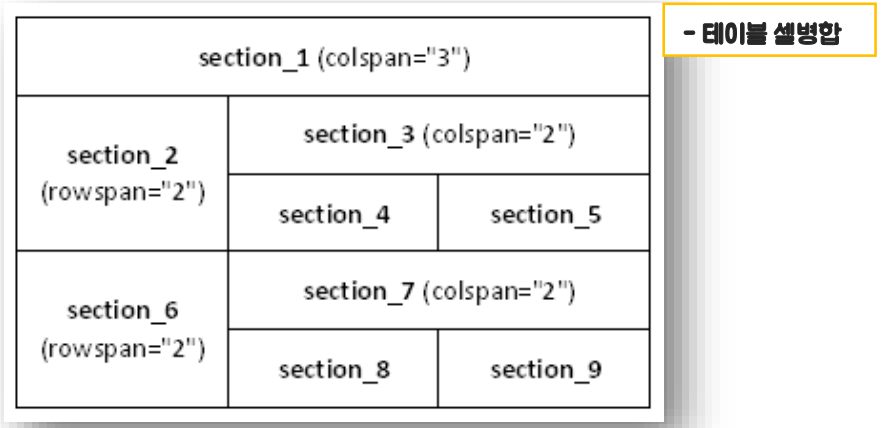
3.<th>

테이블 제목

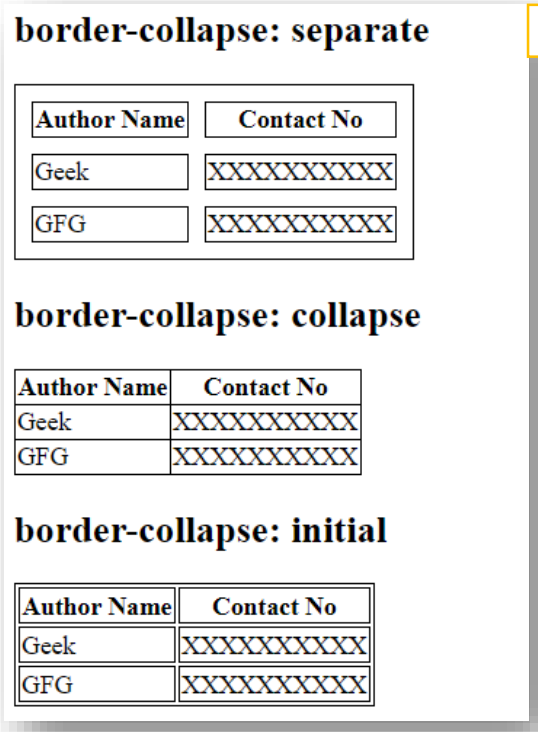
4.<td>

테이블 데이터[셀]

- 테이블 셀병합



- 테이블 테두리 병합



CSS

1.정의

적용방법 1

```
<a href="http://blog.naver.com/runpeaceya"
    style="font-weight: bold; color: blue;">WHAT YOU LOOK</a>
```

적용방법 2

```
<html>
    <head>
        <meta charset="UTF-8" />
        <title>외부 CSS</title>
        <style type="text/css">
            a { font-weight: bold; color: blue; }
        </style>
    </head>
    <body>
        <a href="http://blog.naver.com/runpeaceya">WHAT YOU LOOK</a>
    </body>
</html>
```

적용방법 3

```
//style.css
a { font-weight: bold; color: blue; }

//test.html
<html>
    <head>
        <meta charset="UTF-8" />
        <title>외부 CSS</title>
        <link rel="stylesheet" type="text/css" href="style.css" media="screen" />
    </head>
    <body>
        <a href="http://blog.naver.com/runpeaceya">WHAT YOU LOOK</a>
    </body>
</html>
```

1.정의

CSS : Cascading Style Sheets
1. 주로 HTML , XML 문서에서 사용 [html 꾸미기]
2. 레이아웃과 스타일 정의
3. 확장자 : 파일명.css

2.주석

/* 주석내용 */

3. HTML에서 CSS 적용방법

1. 태그의 style속성 이용
   <태그 style="속성 : 값" >
2. head태그 안에서 css 정의
   <style>
   선택자{ 속성 : 값 ; }
   </style>
3. css 파일 import 하기
   <link href='css파일경로" rel="stylesheet">

2.선택자

명칭	선택자	설명
전체 선택자	* {속성:속성 값;}	페이지에 있는 모든 요소를 대상으로 스타일을 적용한다.
태그 선택자	태그 {속성:속성 값;}	특정 태그에 스타일을 적용한다.
클래스 선택자	.클래스이름 {속성:속성 값;}	특정 클래스에 스타일을 적용한다.
id선택자	#id이름 {속성:속성 값;}	특정 id에 스타일을 적용한다.
하위 선택자	상위요소 하위요소 {속성:속성 값;}	모든 하위 요소에 스타일을 적용한다.
자식 선택자	부모요소 > 자식요소 {속성:속성 값;}	자식 요소에만 스타일을 적용한다.
인접 형제 선택자	요소1 + 요소2 {속성:속성 값;}	첫 번째 동생 요소에 스타일을 적용한다.
형제 선택자	요소1 ~ 요소2 {속성:속성 값;}	형제 요소에 스타일을 적용한다.
그룹 선택자	요소1,요소2 {속성:속성 값;}	여러 요소들에 한번에 묶어서 스타일을 적용한다.

[선택자(selector)]

CSS 선택자

선택자

HTML 요소를 꾸미기 위한 선택

1. ***{ }** : 모든 곳에 적용
2. 태그명**{ }** : 해당 태그에 적용
3. **.class**명**{ }** : 해당 **class**명을 가진 태그에 적용 **[** 복수 적용 **]**
4. **#id{ }** : 해당 **id**명을 가진 태그에 적용 **[** 단일 적용 **]**

***<p style="" >** : 해당 태그에 바로 적용

----- 우선순위 : 속성이 중복일 경우 **------**

style > id > class > 태그 **> ***

----- 강제로 우선 속성 적용하기 : **!important --------**

```
div{ color : red !important; }
```

3.속성 텍스트

1.font-family :

폰트명

2.color

폰트 색상

3.text-decoration

밑줄

4. text-shadow

폰트 그림자

5. text-align

폰트 가로 정렬

6. vertical-alian

폰트 세로 정렬

7. font-szie

폰트 사이즈

8. font-style

폰트 스타일

9. font-weight

폰트 진하게

10. word-spacing

단어 간격(어간)

11. line-height

글자 간격(자간)

12. letter-spacing

줄간격

12. list-style-type

글머리

웹폰트

눈누 : https://noonnu.cc

다운로드 없이 웹주소 으로부터 폰트 가져오기

- 인터넷이 되면 누구든 어디서든 사용가능

1.사용할 웹폰트 코드 복사

웹폰트로 사용 복사하기

```
@font-face {
    font-family: 'SDSamlipho
    src: url('https://cdn.js
    font-weight: normal;
    font-style: normal;
}
```

2.CSS파일에 붙여넣기

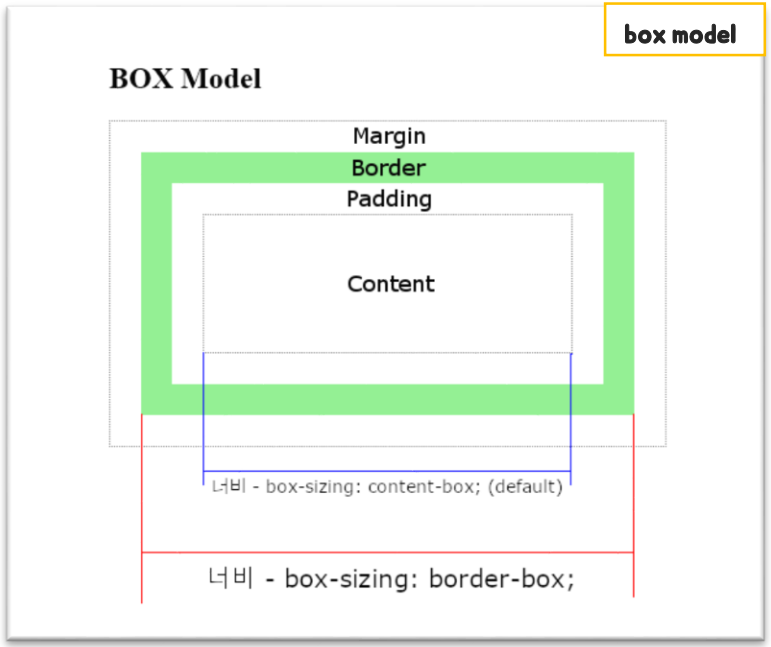
```
@font-face {
    font-family: 'SDSamliphopangche_Outline';
    src: url('https://cdn.jsdelivr.net/gh/projectnoonnu/noonfonts-20-12@1.0/SDSamliphopang
    font-weight: normal;
    font-style: normal;
}
```

3.사용할 선택자에 웹폰트명 작성

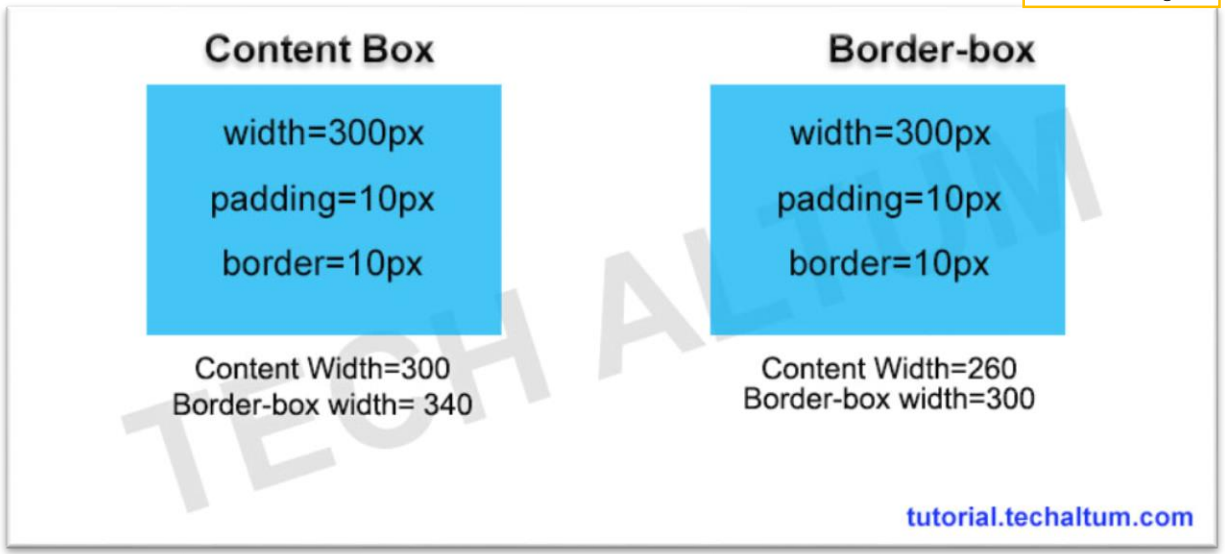
```
body {
    font-family: 'SDSamliphopangche_Outline';
}
```

4.속성 모델

box model



box-sizeing



1.border :

-테두리 위치

border	: 테두리
border-top	: 위 테두리
border-right	: 오른쪽 테두리
border-bottom	: 아래 테두리
border-left	: 왼쪽 테두리

- 선유형

solid	: 실선
dotted	: 점선
double	: 이중선
dashed	: 파선

2.border-collapse :

테두리 병합

3.border-radius :

둥근 모서리

4. box-shadow

박스 그림자

5. box-sizing

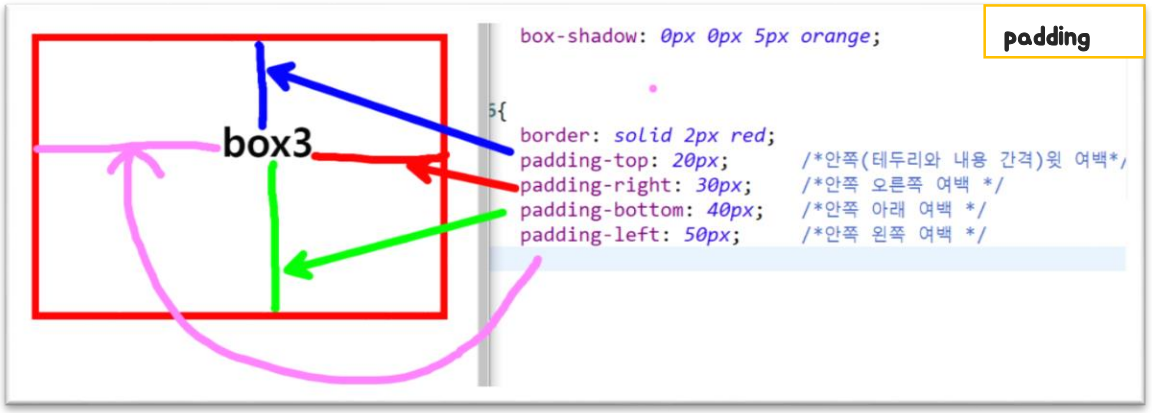
박스 사이즈 기준점

6. width

가로길이

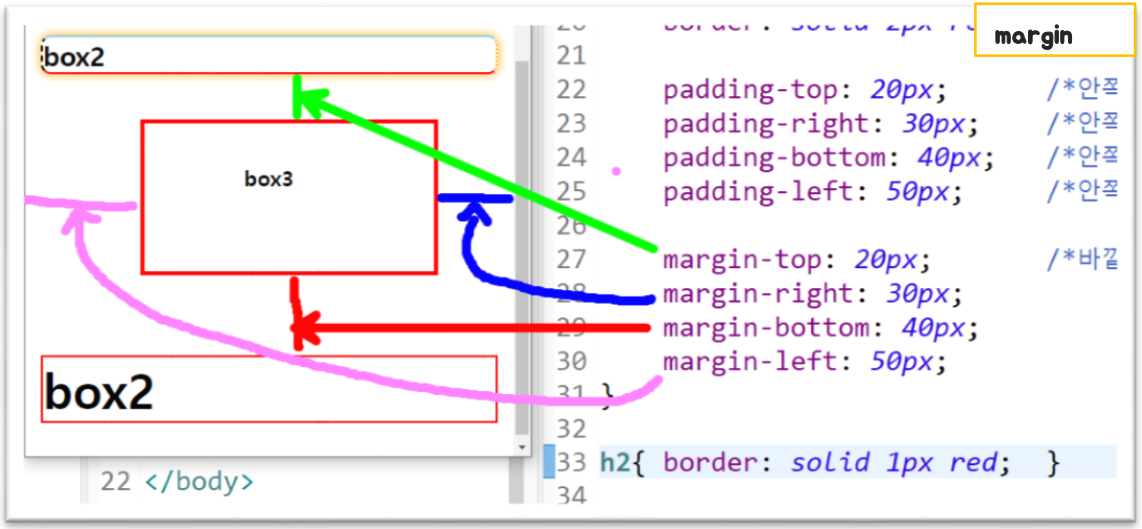
7. height

세로길이



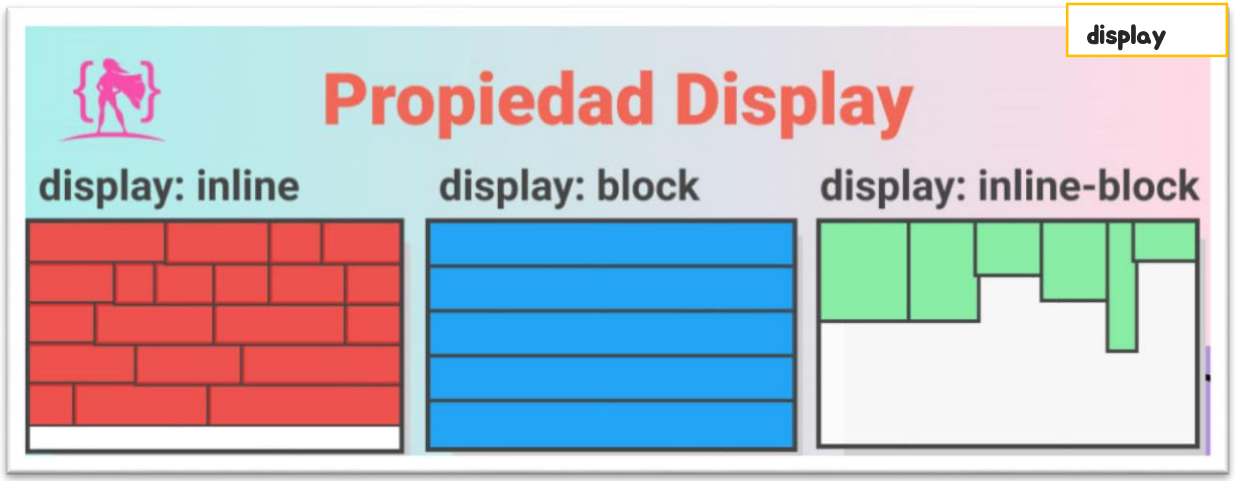
1.padding : 안쪽여백

padding-top : 10px :안쪽 윗 여백
padding-right : 10px :안쪽 오른쪽 여백
padding-bottom : 10px :안쪽 아래 여백
padding-left : 10px :안쪽 왼쪽 여백
padding : 10px :모든 위치 안쪽 여백
padding : 10px 0px :위/아래 왼/오른쪽
padding : 10px 20px 30px 40px :위 오 아 왼



2.margin : 바깥여백

margin-top : 10px :바깥 윗 여백
margin-right : 10px :바깥 오른쪽 여백
margin-bottom : 10px :바깥 아래 여백
margin-left : 10px :바깥 왼쪽 여백
margin : 10px :모든 위치 바깥 여백
margin : 10px 0px :위/아래 왼/오른쪽
margin : 10px 20px 30px 40px :위 오 아 왼



1.display :

display : none　　표시 숨기기
display : block　　한줄 차지 적용 [width , height 적용 O]
기본값으로 사용하는 태그들 : div , h1 , p , ul , ol , li 등

display : inline　　같은 줄에 적용 [width , height 적용 X]
기본값으로 사용하는 태그들 : span, a

display : inline-block inline+block[width , height 적용 O]
기본값으로 사용하는 태그들 : button , input , select 등

RGB color table

CSS 색상

Basic colors:

Color	HTML / CSS Name	Hex Code #RRGGBB	Decimal Code (R,G,B)
	Black	#000000	(0,0,0)
	White	#FFFFFF	(255,255,255)
	Red	#FF0000	(255,0,0)
	Lime	#00FF00	(0,255,0)
	Blue	#0000FF	(0,0,255)
	Yellow	#FFFF00	(255,255,0)
	Cyan / Aqua	#00FFFF	(0,255,255)
	Magenta / Fuchsia	#FF00FF	(255,0,255)
	Silver	#C0C0C0	(192,192,192)
	Gray	#808080	(128,128,128)
	Maroon	#800000	(128,0,0)
	Olive	#808000	(128,128,0)
	Green	#008000	(0,128,0)
	Purple	#800080	(128,0,128)
	Teal	#008080	(0,128,128)
	Navy	#000080	(0,0,128)

색상

Css에서 색상을 표기하는 3가지 방법

color : 폰트 색상
background-color : 배경 색상

색상 표현 방법
1. RGB : 빨강초록파랑 혼합한 색상 0~255
rgb(255, 255, 255)

2. Hex color : 16진수 이용한 색상 표현
#FFFFFF

3. 키워드 : CSS 키워드 [140개 지원]
white

참고 : https://htmlcolorcodes.com/color-names/

4.속성 레이아웃

div vs 레이아웃 마크업

1. 헤더
  <header> </header>

2. 메뉴
  <nav> </nav>

3. 본문
  <section> </section>

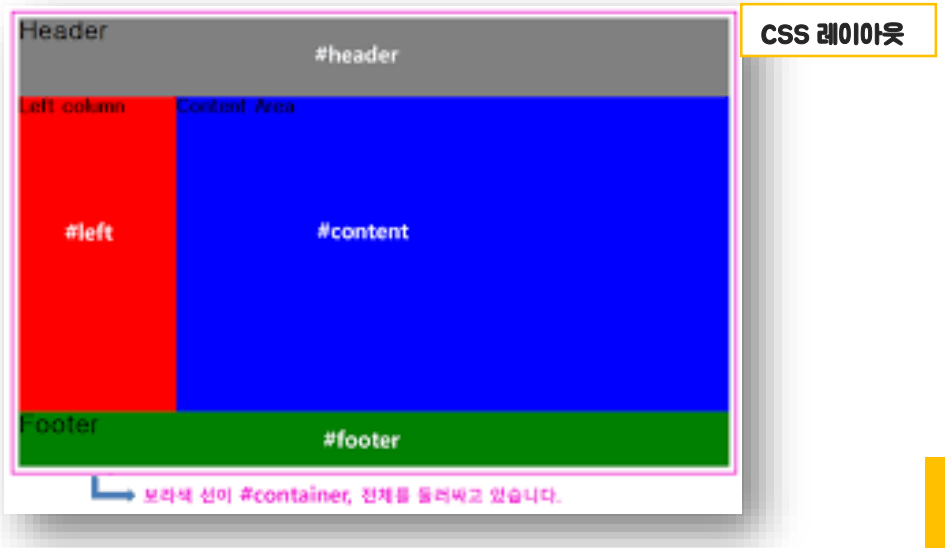
4.사이드바
  <aside> </aside>

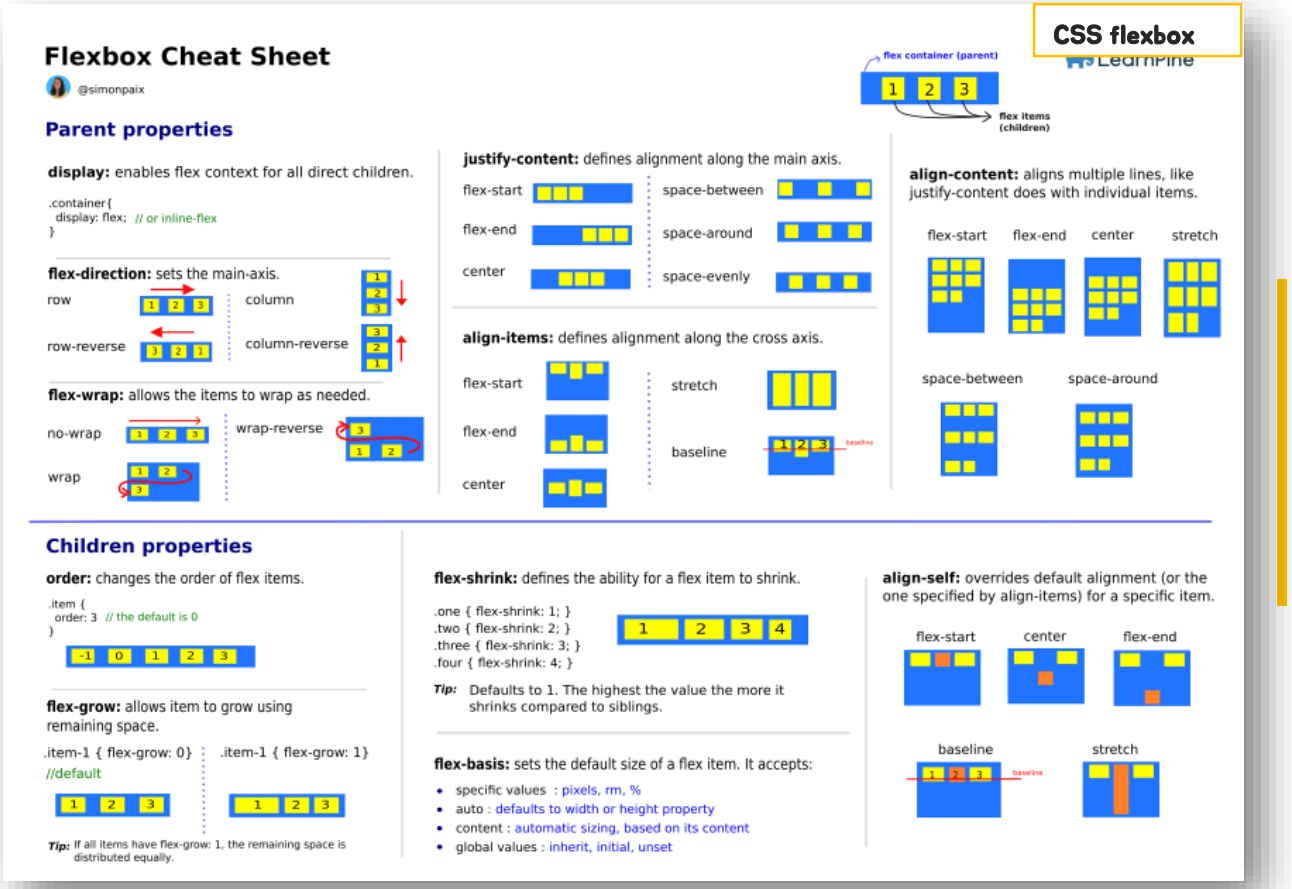
5. 푸터
  <footer> </footer>

레이아웃

제한된 공간에서 효율적인 배치

div vs span 주로 사용
- 1. 일반적인 마크업들은 기본 css 속성 존재
- 2. div 와 span display 기본 속성 존재
- 3. HTML 5 부터 레이아웃 마크업이 존재

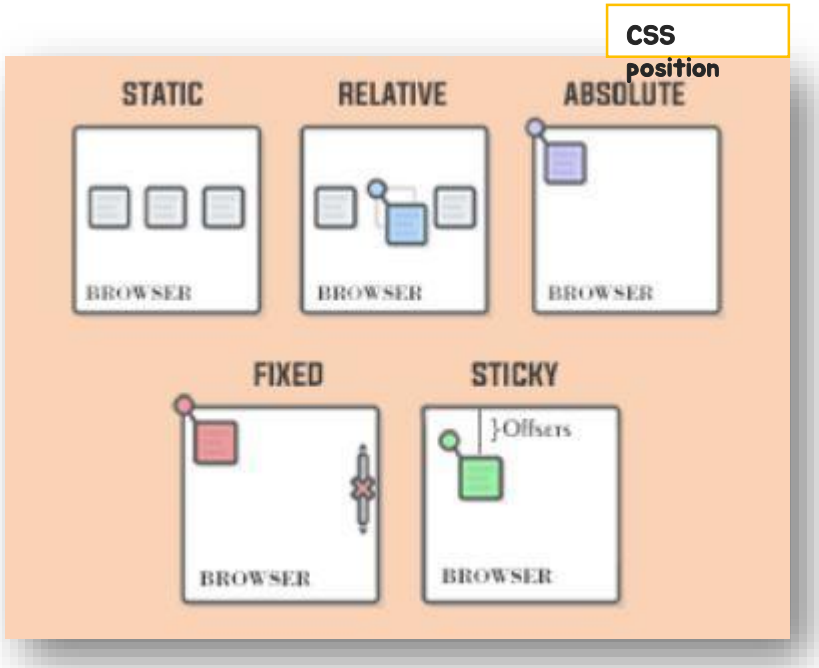




1.flex :

1. display : flex [flex 사용할 요소들의 상위요소 적용]
3. justify-content: 가로 정렬
4. flex-direction : 배치방법
   1. row : 가로 배치 [기본값]
   2. column : 열 배치
   3. row-reverse : 가로 배치 [반대로]
   4. column-reverse: 열 배치 [반대로]
5. flex-wrap:
   1. nowrap : [기본값]
   2. wrap : 자동 줄바꿈

6.align-items: : 세로 정렬
- flex-start : 위쪽 정렬
- center : 가운데 정렬
- flex-end : 아래쪽 정렬
- stretch : heigth/width 없는 요소의 자동으로 너비 설정
- baseline : 내용 기준 정렬



1.position :

position: static : [기본값] 작성 순서대로 자동 배치 [위치 선택 불가]
position: relative : 상대경로[본래 위치 기준]
position: absolute : 절대경로[상위 요소 기준]
- 상위 요소 position: relative
position : fixed : 화면 위치 고정
position : sticky : 해당 요소가 화면에서 나갔을때 위치 배치 고정

- 위치 선택 : border 기준
top : 위쪽부터 + : 아래로 - 위로
left : 왼쪽부터 + : 오른쪽으로 - 왼쪽으로
right : 오른쪽부터+ : 왼쪽으로 - 오른쪽으로
botto : 아래부터 + : 위로 - 아래로

- 위치가 겹칠때의 배치 순서 :
z-index : 0 [기본값]
번호가 높은 순서대로 위에 배치

요소1	요소2	요소3
z-index:3	z-index: 생략	z-index:-1

배치 순서 : 요소1 > 요소2 > 요소3

- 투명 색상
1. opacity : 투명도
[1 : 100% ~ 0 : 0% 0.5 : 50%;]
2. background-color : transparent : [기본값] 색상없음 [투명색]

가상클래스

CSS 가상클래스

종류	표시 방법	설명	CSS level
가상클래스	E:link	방문 전 링크	1
	E:visited	방문 후 링크	1
	E:active	마우스를 클릭할 때	1
	E:hover	마우스 오버했을 때	1
	E:focus	요소에 초점이 맞춰질 때	1&2

JS

1.정의

Ecma International의 프로토타입 기반의 프로그래밍 언어로, 스크립트 언어에 해당된다. 특수한 목적이 아닌 이상 모든 웹 브라우저에 인터프리터가 내장되어 있다. 오늘날 HTML, CSS와 함께 웹을 구성하는 요소 중 하나다. HTML이 웹 페이지의 기본 구조를 담당하고, CSS가 디자인을 담당한다면 JavaScript는 클라이언트 단에서 웹 페이지가 동작하는 것을 담당한다.[1] 웹 페이지를 자동차에 비유하자면, HTML은 자동차의 뼈대, CSS는 자동차의 외관, JavaScript는 자동차의 동력원인 엔진이라고 볼 수 있다.

썬 마이크로시스템즈(현재 오라클)에서 개발한 Java와는 별 관계가 없는 언어이다. 이름이 비슷하다고 같은 언어가 아니다. 사람들이 흔히 헷갈리는 부분 중 하나라 Java의 소유자인 오라클에서도 아니라고 강조한다.[2] 실질적인 구동 방식도 Java Virtual Machine을 이용해서 돌리는 Java와, 브라우저 내에 스크립트 엔진(인터프리터)이 존재하는 JavaScript는 완전히 다르다. 햄이랑 햄스터가 다르고 파와 파슬리가 다르며, 인도가 인도네시아와 다르듯 심지어는 웹 서버용 파생 규격도 다르다.

1.정의

JS : JavaScript
- HTML 과 CSS 언어는 주로 정적인 역할
- JS는 HTML 에 이벤트[행동] 실행하는 동적인 역할
- 확장자 : 파일명.js

2.주석

// : 한줄 주석 /* 여러줄 주석 */

3. HTML에서 JS 적용방법

1.
```
<script>
        alert('하하');
</script>
```
2.
```
<div onclick=" alert('하하') "> </div>
```
3.
```
js파일 생성
<script src='js파일경로'> </script>
```

2.출력/입력

```
// 1. 출력 [ console.log( 출력할 데이터/값 ) ]
console.log('Hello World') // ' ' 안에 있는 데이터는 문자처리
//console.log(Hello World) // !문자처리 생략시 오류 발생
console.log("Hello World") // " " 안에 있는 데이터는 문자처리
console.log( 200 ) // 숫자는 문자처리 안함
console.log( "200" ) // 숫자에 문자처리 하면 문자로 출력
console.log( true ) // 논리는 문자처리 안함
// 2. 출력 [ alert( 출력할 데이터/값 ) ]
alert('Hello World2')
alert("Hello World22")
alert( 200+200 )
alert( true )
// 3. 제어문자
console.log('안녕하세요\nJS 처음입니다.')
console.log('안녕하세요\tJS 처음입니다.')
console.log('안녕하세요\\JS 처음입니다.')
console.log('안녕하세요\'JS처음입니다\'')
console.log('안녕하세요\"JS처음입니다\"')
```

```
//1. 입력메시지에 대한 확인/취소 버튼 결과를 변수에 저장
let result1 = confirm('실행하시겠습니까?')
console.log( '결과1 : ' + result1 )
//2. 입력메시지에 입력된 데이터를 변수에 저장
let result2 = prompt('데이터 입력')
console.log( '결과2 : ' + result2 )
```

1.데이터 유형

문자 : ' ' or " "
논리 : true or false
숫자 : 300

2. 출력

1. console.log(메시지) : 콘솔에 출력
2. alert(메시지) : 알람메시지 형식 출력
3. document.write(메시지) : HTML body 출력

3. 입력

1. confirm(메시지) : 확인(true)/취소(false) 버튼 입력
2. prompt(메시지) : 입력상자에 데이터 입력

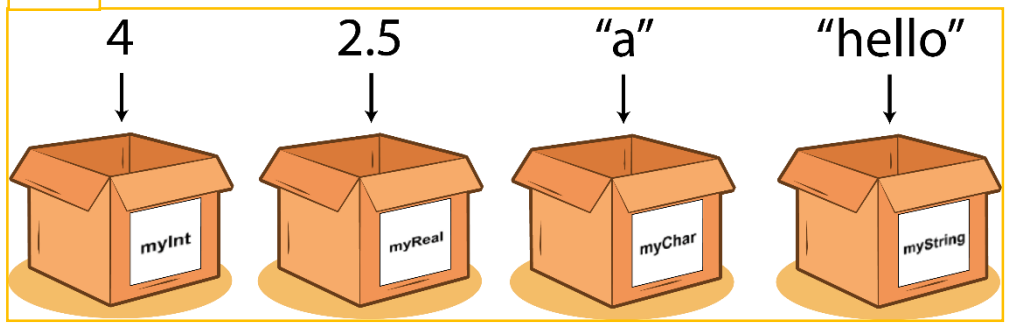
문자형 : let 변수명 = prompt(메시지)
숫자형 : let 변수명 = Number(prompt(메시지))
논리형 : let 변수명 = Boolean(prompt(메시지))

4. 제어문자[이스케이프문자]

1. ₩n : 줄바꿈
2. ₩t : 들여쓰기
3. ₩₩ : ₩ [백슬래시] 출력
4. ₩' : ' 출력
5. ₩" : " 출력

3.변수와상수

변수




```
// 1. 변수 선언
let value1 = 10 // 'value1' 이라는 변수 상자에 10 대입했
// 2. 변수 값 호출
console.log( '변수값 : ' + value1 ) // 'value1' 이라는 변수 상자에 값 을 호출
// 3. 변수 값 변경
// let value1 = 20 // 'value1' 이라는 변수명이 존재하기때문에 오류 발생
value1 = 20 // 'value1' 이라는 변수 상자에 새로운 값 대입했다.
console.log( '변수값 수정후 : ' + value1 )
// 4. 변수 연산
console.log( '변수값 더하기 5 : ' + value1+5 ) // 문자열 + 숫자 + 숫자
console.log( '변수값 더하기 5 : ' + (value1+5) ) // 문자열 + ( 숫자 + 숫자 )
let value2 = 30
let result = value1 + value2
console.log( '변수들 간에 연산 : ' + result )
```

```
// 1. 상수 선언
const data1 = 10
console.log( '상수값 : ' + data1 )
//data1 = 20 //!! : 변경 불가 [ 상수 ]
const data2 = 20
const result2 = data1 + data2
console.log( '상수들 간에 연산 : ' + result2 )
```

1.변수와 상수

- 데이터 1개 저장할수 있는 메모리 공간
- 준비물 :
    1. 자료형키워드
    2. 변수/상수 명
    3. 저장할 데이터
    4. = : 대입연산자 [오른쪽 데이터를 왼쪽에 넣기]

변수 : 변하는 수
    1. Let 키워드 사용
    2. 형태 : let 변수명 = 데이터
    3. 대입 후 에 변경 가능

상수 : 고정된 수
    1. Const 키워드 사용
    2. 형태 : const 변수명 = 데이터
    3. 대입 후 에 변경 불가능

선언 방법
    1. let or const
    2. 변수명 정의[만들기]
    3. = 대입연산자
    4. 데이터

변수 호출
    1. 변수명 이용한 호출

변수 값 변경
    1. 변수명 = 새로운 데이터

4.연산자

```
//1.산술연산자
console.log( 10+3 ) // 숫자+숫자 => 숫자
console.log( '더하기 : ' + 10+3 ) // 문자+숫자 => 문자 + 숫자 => 문자
console.log( '더하기 : ' + (10+3) ) // (숫자+숫자) => 숫자 + 문자 => 문자
console.log( '빼기 : ' + (10-3) )
console.log( '곱하기 : ' + (10*3) )
console.log( '나누기 : ' + (10/3) )
console.log( '나머지 : ' + (10%3) ) // 몫 제외한 나머지 !!
//2.비교연산자
console.log( '초과 : ' + (10>3) ) //t
console.log( '미만 : ' + (10<3) ) //f
console.log( '이상 : ' + (10>=3) ) //t
console.log( '이하 : ' + (10<=3) ) //f
console.log( '같다 : ' + (10==3) ) //f
console.log( '같지않다 : ' + (10!=3) ) //t
//3.관계연산자
console.log( '이면서 : ' + ( 10>3 && 10>5 ) ) // T and T -> T / T and F -> F
console.log( '이거나 : ' + ( 10>3 || 3>5 ) ) // T or F -> T
console.log( '부정 : ' + !( 10>3) ) // T->F / F->T
```

```
// 4. 대입연산자
let data1 = 10 // 1. = 대입 [ 선언과 동시에 대입 = 초기화 ]
data1 += 2 // vs data1 = data1 + 2 // 1. 10+2 --> 12 // 2. data1 = 12
console.log( ' += 대입 후 : '+ data1 ) // 10+2 -> 12
data1 -= 5; console.log( ' -= 대입 후 : ' + data1 ) // 12-5 -> 7
data1 *= 2; console.log( ' *= 대입 후 : ' + data1 ) // 7 * 2 -> 14
data1 /= 3; console.log( ' /= 대입 후 : ' + data1 )
data1 %= 3; console.log( ' %= 대입 후 : ' + data1 )
// 5. 증감연산자 [ data++ , data+=1 , data = data+1 ]
let data2 = 10;
console.log( '변수 값 : ' + data2 ) // 10
console.log( '변수++ : ' + (data2++) ) // 10 : 출력후 증가
console.log( '변수 값 : ' + data2 ) // 확인 : 11
console.log( '++변수 : ' + (++data2) ) // 12 : 출력전 증가
console.log( '변수-- : ' + ( data2-- ) ) // 12 : 출력후 감소
console.log( '변수 값 : ' + data2 ) // 확인 : 11
console.log( '--변수 : ' + (--data2) ) // 10 : 출력전 감소
// 6. 삼항연산자 [ 조건 ? 참 : 거짓 ]
let age = 38;
let 나이대 = age<=19 ? '청소년' : '성인' // 조건[ 38<=19 ]이 false 니까 '성인' 대입
console.log( ' 38의 나이대 : ' + 나이대 )
```

1. 연산자

1.산술연산자 :

+더하기 -빼기
*곱하기 /나누기
%나머지

2.비교연산자 :

>초과 <미만
>=이상 <=이하
== 데이터만 같다
=== 데이터/자료형 같다.
!= 데이터만 같지않다
!== 데이터/자료형 같지않다.

3. 관계연산자 :

&& 이면서
|| 이거나
! 부정

4. 대입연산자 :

= 오른쪽데이터를 왼쪽에 대입
+= 오른쪽데이터를 왼쪽에 더한 후에 대입
-= *= /= %=

5. 증감연산자 :

변수++ [후위 1증가] 변수-- [후위 1감소]
++변수 [선위 1증가] --변수 [선위 1감소]

6.삼항연산자 :

조건 ? 참 : 거짓
조건1 ? 참1 : 조건2 ? 참2 : 조건3 ? 참3 : 거짓

*7.연결연산자 :

'문자' + 숫자 => 문자
숫자 + 숫자 => 숫자
'문자' + (계산식) => '문자'+계산식결과

8. 자주 사용되는 공식

1. 짝수 / 홀수 찾기
   - 값 % 2 == 0 : 나머지가 0 이면 짝수
   - 값 % 2 == 1 : 나머지가 1 이면 홀수
2. 배수 찾기
   - 값 % 수 == 0 : 나머지가 0 이면 수 는 해당 값의 배수

5.배열

```
// 1. 배열 선언 : [] 안에 , 구분해서 여러개의 자료 입력한다.
let 배열명 = [ '유재석' , '강호동' , '신동엽' , '김현수' ]
// 2. 배열 호출
console.log( 배열명 )
document.write( 배열명 )
document.write('<h3>'+배열명+'</h3>') // * HTML 마크업은 JS내에서는 문자열 처리
// 3. 배열내 특정 요소만 호출 : 배열명[인덱스]
console.log( 배열명[0] + 배열명[1] + 배열명[2] + 배열명[3] )
document.write('<ol>')
document.write('<li>'+배열명[0]+'</li>') // 변수/배열 문자열 처리X // HTML 문자열처리 O
document.write('<li>'+배열명[1]+'</li>') // 변수/배열 문자열 처리X // HTML 문자열처리 O
document.write('<li>'+배열명[2]+'</li>') // 변수/배열 문자열 처리X // HTML 문자열처리 O
document.write('<li>'+배열명[3]+'</li>') // 변수/배열 문자열 처리X // HTML 문자열처리 O
document.write('</ol>')
// 4. 배열의 길이
console.log( 배열명.length +"명" )
document.write('<p>현재 인원수 : '+배열명.length+'명</p>')
// 5. 배열의 요소 추가
배열명.push('전현무')
console.log( 배열명 )
// 6. 배열의 특정 요소 제거
배열명.splice( 0 , 1 );
console.log( 배열명 ) // ['유재석', '강호동', '신동엽', '김현수', '전현무']
// * 한줄에 두개 이상의 명령어 작성시에는 ; 구분하기
// 7. 데이터로 인덱스 찾기
let sindex = 배열명.indexOf('신동엽');
console.log( '신동엽 인덱스 : ' + sindex )
// 8. 데이터로 요소 삭제
배열명.splice( 배열명.indexOf('신동엽') , 1 );
console.log( 배열명 ) // ['강호동', '김현수', '전현무']
// 9. 배열내 요소 데이터 변경
배열명.splice( 0 , 1 , '서장훈');
console.log(배열명) // ['서장훈', '김현수', '전현무']
// 10. 배열내 요소 사이[ 서장훈 과 김현수 사이 ] 요소 추가
배열명.splice( 1 , 0 , '유재석' )
console.log( 배열명 ) // ['서장훈', '유재석', '김현수', '전현무']
```

1.배열

1. 배열 선언
   let 배열명 = [요소1 , 요소2 , 요소3]
   const 배열명 = [요소1 , 요소2 , 요소3]
2. 배열 호출
   1.전체 : 배열명
   2.특정요소 : 배열명[인덱스번호]

3. 인덱스 : 요소의 저장된 순서번호 = 식별역할 / [0] 부터 시작

4. 배열의 길이 : 배열명.length

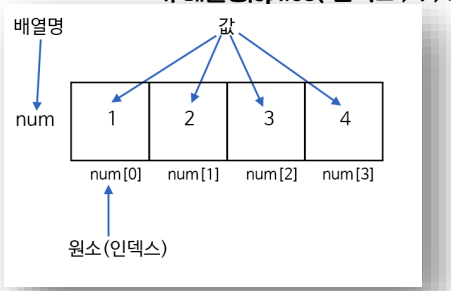
5. 배열내 요소 추가
   1.배열명.push(새로운요소) : 마지막 인덱스 뒤에 추가
   2. 배열명.splice(인덱스 , 0 , 새로운요소) : 해당 인덱스위치에 새로운 요소 추가

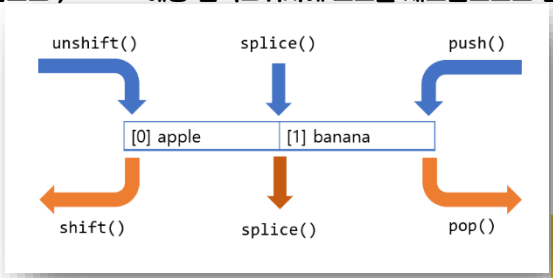
6. 배열내 요소 삭제
   1. 배열명.splice(0) : 모든 요소 삭제 / 배열 초기화
   2. 배열명.splice(인덱스 , 1) : 해당 인덱스 1개 삭제
   3. 배열명.splice(배열명.indexOf(데이터) , 1) : 해당 데이터의 인덱스 1개 삭제

7. 배열내 데이터 인덱스 찾기
   1. 배열명.indexOf(데이터) : 해당 데이터의 인덱스 찾기
   [없으면 -1 / 존재하면 찾은해당인덱스번호 반환]

8. 배열내 데이터 존재여부 확인
   1. 배열명.includes(데이터) : 해당 데이터의 존재여부 확인
   [존재하면 true / 없으면 false 반환]

9. 배열내 요소 데이터 변경
   1. 배열명.splice(인덱스 , 1 , 새로운요소) : 해당 인덱스위치에 요소를 새로운요소로 변경





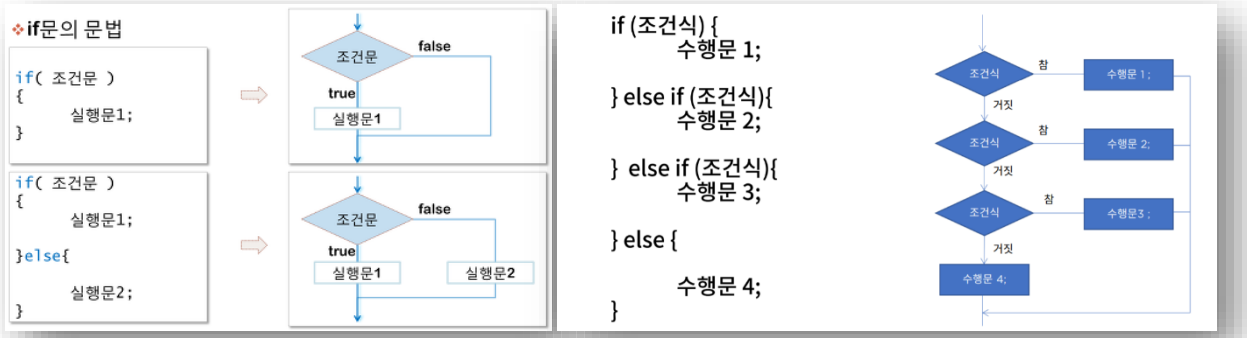
6.제어문

```
// 1. if 형태1
if( 10 > 3 ) console.log('[참1] 10이 더 크다. ')
if( 10 > 20 ) console.log( '[참2] 10이 더 크다.')
if( 10 > 20 ); console.log( '[참3] 10이 더 크다. ') // [ X ]

// 2. if 형태2
if( 10 > 3 ) console.log('참1'); console.log('1.10이 더 크다.'); // [x] 2번째
console if랑 상관없이 무조건 실행
if( 10 > 20 ) console.log('참2'); console.log('2.10이 더 크다. '); // [x] 2번째
console if랑 상관없이 무조건 실행
if( 10 > 20 ) { console.log('참3'); console.log('3.10이 더 크다.'); } // [O]

// 3. if 형태3
if( 10 > 3 ){ console.log('[참1] 10 더 크다.'); }
else{ console.log('[거짓1] 10 더 작다.'); }

// 4. if 형태4
if( 10 >= 20 ){ console.log( '[참1] 10이 20보다 이상.') } // 만약에 10>=20 이상이면
else if( 10 >= 15 ) { console.log( '[참2] 10이 15보다 이상.') } // 아니면서 만약에
10>=15 이상이면
else if( 10 >= 10 ) { console.log( '[참3] 10이 10보다 이상') } // 아니면서 만약에
10>=10 이상이면
else{ console.log('[거짓] 10이 10 미만이다. ') } // 그외
console.log('[거짓] 10이 10 미만이다. ')
```



-if 제어문

1. 경우의수/조건 판단 할때 사용 [알고리즘 흐름 제어]
2. 형태

1. if(조건) 실행문

2.if(조건) { 실행문; 실행문; }

3. 조건이 참/거짓일때

```
if( 조건 ) { T실행문; }
else{ F실행문; }
```

4. 조건이 다수일때

```
if( 조건1 ) { T1실행문; }
else if( 조건2 ) { T2실행문; }
else if( 조건3 ) { T3실행문; }
else { F실행문; }
```

5. if중첩

```
if( 조건 ) {
        if( 조건 ) {

        }else{

}
}else{
        if( 조건 ) {

        }else{

        }

}
```

7.반복문

```
// 1. for문
for( let j = 0 ; j<10 ; j++ ){ // for s
// j는 0부터 10미만[9] 까지 1씩증가 하면서 반복 = 10회 반복
console.log( '실행문3 : ' + j )
}// for e
// 2. 배열과 for문
let 과일상자 = [ '사과' , '포도' , '딸기'] // * 데이터 3개를 저장하는 배열 선언
// 1. 0부터 마지막인덱스까지의 J 반복해서 배열명[인덱스=J] 요소 호출
for( let j = 0 ; j < 과일상자.length ; j++ ){ // for s
// j는 0부터 배열의길이[3] 미만 까지 1씩 증가 하면서 반복 = 0 1 2 -> 3회
// 인덱스[ 0부터 시작 ] === 길이 [ 1부터 시작 ]
console.log( j + "번째 요소 :"+ 과일상자[j] )
} // for e
// 2. of 키워드 : 기준으로 오른쪽에 있는 배열/리스트내 요소 값를 하나씩 왼쪽 변수에 반복 대입
for( let j of 과일상자 ){ // for s
console.log( "요소의 데이터 : " + j )
}// for e
// 3. in 키워드 : 기준으로 오른쪽 있는 배열/리스트내 요소 인덱스를 하나씩 왼   변수에 반복 대입
for( let j in 과일상자 ){
console.log( "요소의 인덱스 : " + j )
}
```

```
for(let i=0; array.length; i++) {
    ...;
}
```

For 기본 문법

VS

```
for(let key in map) {
    ...;
}

for(let element of array) {
    ...;
}
```

For 변형된 문법
(for..in / for..of)

VS

```
array.forEach(function(element){
    ...;
})

array.map(function(element){
    return ...;
})
```

For 변형된 문법
(forEach / map)

- 반복문

* 1.for(1.초기값 ; 2.조건식 ; 3.증감식){ 4.실행문 }

1. 초기값 : 반복 변수의 시작값
2. 조건식 : 반복 실행되는 조건 [true이면 실행 / false이면 미실행]
3. 증감식 : 반복되는 변수의 증가/감소 단위
4. 실행문 : 반복되는 실행 코드

* 2. 배열내 요소(데이터)들을 하나씩 반복변수에 대입
for(반복변수 of 배열){ 실행문; }

* 3. 배열내 요소의 인덱스 들을 하나씩 반복변수에 대입
for(반복변수 in 배열) { 실행문; }

* 4. 배열내 요소(데이터,인덱스) 들을 하나씩 반복변수에 대입
배열.forEach((object , index) => { 실행문; })

* 5. 배열내 요소(데이터,인덱스) 들을 하나씩 반복변수에 대입후 retrun 값을 새로운 배열 생성
let 새로운배열 = 배열.map((object , index)=> { 실행문; retrun 값; })

8.함수

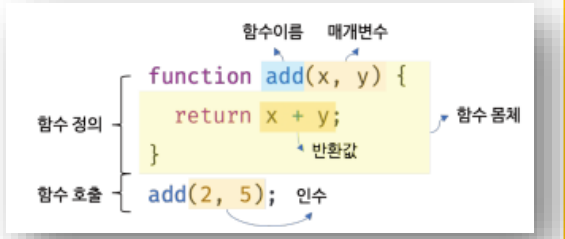
```
// 1. 인수X 반환X 함수
function 함수1(){ // f s
        // function : 함수 선언시 사용되는 키워드
        // 함수1 : 함수이름[식별자] 동일한파일내에서는 중복이름 불가능
        // ( ) : 인수 정의하는 곳
        // { } : 함수가 호출되면 실행 되는 구역
        alert('함수1 실행 됨')
} // f e

// 1. 함수 호출
함수1()

// 2. 인수O 반환O 함수
function 함수2( x , y ){ // f s
        // ( x , y ) : 해당 함수를 호출시 인수[ x와 y : 이름 아무거나 ] 2개를 받는 함수
        alert('함수2 실행 됨')
        return x + y;
} // f e
let result = 함수2( 3 , 5 ) // 반환[return] 값을 변수에 저장
alert( '함수2 실행후 보내준 값 :' + result )

// 3. 인수O 반환X 함수
function 함수3( x , y , z ){ // f s
        let result = x + y + z
        alert( '함수3 실행 됨 : ' + result )
} // f e
함수3( 3 , 5 , 7 )

// 4. 인수X 반환O 함수
function 함수4( ){ // f s
        let result = 3 + 5 + 8
        return result;
} // f e
let result2 = 함수4()
alert( '함수4 실행후 보내준 값 :' + result2 )
```



- 함수 [함 : 상자 수 : 숫자/코드]
  - 상자{ } 안에 들어있는 수/코드 -> 상자 미리 넣어둔 수/코드
  -> 미리 정의된 수/코드

1. 역할
   1. 재활용성 : 미리 정의된 코드를 반복적으로 사용
   2. 메모리효율성 : { } 에서만 메모리 할당
      '{' 시작되면 실행 필요한 메모리 할당
      '}' 끝나면 함수내 사용된 메모리 초기화
   3. 인수 와 반환값 : 인수에 따른 서로 다른 결과
      1. 인수[매개변수] : 함수 안으로 들어가는 수[함수를 호출했던 위치에서]
      2. 반환 : 함수 밖으로 나가는수[함수를 호출했던 위치로]
2. 정의
   function 함수명(x , y){ return x+y }
3. 호출
   * 미리 정의된 함수명 과 인수를 동일하게 작성한다.
   함수명(3 , 5)
   let 결과 = 더하기함수(3 , 5)

4.함수를 만들어야 하는 경우의수
   1. 코드 작성하는데 반복적인 코드 있을경우
   2. 여러 개발자들이 같이 사용해야 되는 경우 [console.log( )]
   3. 메모리를 영구저장없이 잠시 사용후 메모리 초기화 해야 되는 경우

5.형태
   1. 인수O 반환O 있는 함수
   let 결과 = 함수명(3 , 5) function(x , y) { return x+y }

   2. 인수O 반환X 있는 함수
   함수명(3 , 5) function(x , y) { x+y }
   * 해당 위치에서는 결과를 알수없다.

   3. 인수X 반환O 있는 함수
   let 결과 = 함수명( ) function( ) { return value; }

   4. 인수X 반환X 있는 함수
   함수명( ) function( ) { }

9.객체

```
// 1. 객체의 선언 : 서로 다른 데이터유형들 간의 식별가능
let 객체1 = { 아이디 : 'qweqwe' , 비밀번호 : '123' , 이름 : '유재석' }
console.log( 객체1 ) // 객체호출

// 2. 객체의 호출
console.log( 객체1 ) // 객체의 모든 정보 호출
console.log( '객체내 아이디 속성의 값 : ' + 객체1.아이디 ) // 객체내 '아이디' 속성의 값 호출
console.log( '객체내 비밀번호 속성의 값 : ' + 객체1.비밀번호 )
console.log( '객체내 이름 속성의 값 : ' + 객체1.이름 )

// 3. 객체의 속성 추가
객체1.주소 = '안산시 상록구'; console.log( 객체1 );
객체1.연락처 = '010-4444-4444'; console.log( 객체1 );

// 4. 객체의 속성 제거
delete 객체1.주소; console.log( 객체1 );

// 5. 속성의 값 변경
객체1.비밀번호 = '789'; console.log( 객체1 );

// 7. 배열과 객체 관계 [ 배열안에 객체저장 가능 / 객체 안에 배열 저장 가능 ]

// 1. 배열안에 여러개 객체 저장 [!! 동일한 객체의 유형 ]
let 게시물1 = { 제목 : '제목1입니다.' , 내용 : '내용입니다1.' }
let 게시물2 = { 제목 : '제목2입니다.' , 내용 : '내용입니다2.' }
let 게시물배열 = [ 게시물1 , 게시물2 ]
// 2. 객체안에 배열 속성 저장
let 공지사항 = {
        게시판이름 :'공지사항' ,
        게시물목록 : 게시물배열
}
```

객체[object]

1. 정의 : 속성과 함수들의 집합
   1. 속성
      속성명 : 데이터 , 속성명 : function () { }

2. 형태 let 객체명 = { }

3. 선언 let 객체명 = { 속성명 : 데이터 , 속성명 : 변수 , 속성명 : 함수 , 속성명 = 배열 }

4. 호출
   1.객체정보 호출 : 객체명
   2.객체내 속성 호출 : 객체명.속성명

5.속성추가 : 객체명.새로운속성명 = 데이터

6.속성삭제 : delete 객체명.삭제할속성명

7.속성의 데이터변경 : 객체명.속성명 = 새로운데이터

8.배열과 객체 관계
   1. let 객체배열 = [{ } , { } , { } , { } , { }]
   2.let 객체 = { 게시물목록 : [ ] }

문서 객체[DOMobject : document]

1. document.body.innerHTML = 데이터 : body마크업에 html형식으로 대입

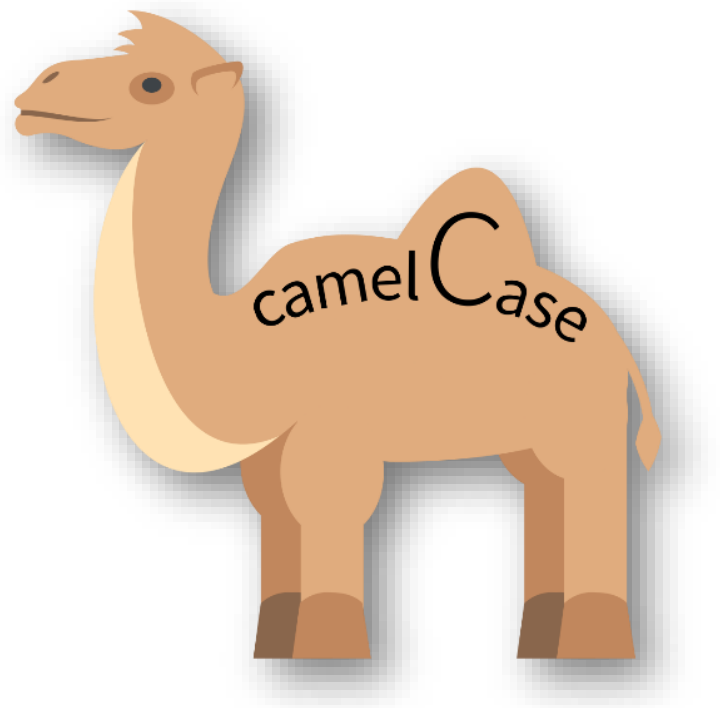
2. 마크업 객체화
   *
     ('div') ('.div') ('#div')
     let html변수명 = document.querySelector('마크업명')
     let html변수명= document.querySelector('.class명')
     let html변수명= document.querySelector('#id명')
   let 객체명 = document.querySelector('식별자') : 해당하는 식별자의 마크업을 1개 객체 반환
   let 객체배열명 = document.querySelectorAll('식별자') : 해당하는 식별자의 모든 마크업 객체를 배열로 반환

3. DOM 속성
   1. 객체명.value : input, select , textarea 에 입력된 값 가져오기
   2. 객체명.innerHTML = 'HTML형식 : div , span , table 등 에서 <마크업> {이자리추가} </마크업>
   3. DOM객체명.classList.add('클래스명') : 해당 DOM객체의 마크업에 class 추가
        -> div.classList.add('wrap') ->
   4. DOM객체명.classList.remove('클래스명') : 해당 DOM객체의 마크업에 class 제거
        -> div.classList.remove('wrap') ->

4.객체명 .addEventListener('이벤트' , (e) -> { })
   * e(event) : 이벤트 실행 결과 정보객체 인수 받음
   1. DOMContentLoaded : html 열렸을때
   2. click : 클릭했을때
   3. keyup : 키보드 키를 떼었을때
   4. keydown : 키보드 키를 눌렸을때
      e.keyCode : 키 코드번호
      e.key : 키 이름
   5. change : 값이 변경 되었을때

5. 객체명.style
   * 카멜표시법 : justify-content -> justifyContent
   객체명.style.justifyContent = 'center'
   객체명.style.fontSize = '13px'
   객체명.style.left = 변수+'px

속성	속성 이름
태그 이름	tagName
id	id
class 목록	classList
class 문자열	className
속성 객체	attributes
스타일 객체	style
엘리먼트에 담긴 내용	innerHTML, innerText, textContent
form 입력 값	value
자식 엘리먼트	children
부모 엘리먼트	parentElement
자식 노드	childNodes
data-* 속성에 담긴 값	dataset
이벤트	onclick, onmouseover, onkeyup 등
좌표 정보 (기준점에 따라 다름)	offsetTop, offsetLeft scrollTop, scrollLeft clientTop, clientLeft
크기 정보 (기준점에 따라 다름)	offsetWidth, offsetHeight scrollWidth, scrollHeight clientWidth, clientHeight



카멜표시법

카멜 표기법(Camel case **카멜 케이스**[1]) **또는 낙타 표기법은 프로그래밍에서 파일, 변수, 함수 등 대상의 이름을 띄어쓰기 없이 짓기 위하여 따르는 관례인 네이밍컨벤션**(Naming convention)**의 하나다.**
단어 전체적으로 소문자를 사용하지만, 맨 첫 글자를 제외한 각 합성어의 첫 글자만 대문자로 표기한다.
합성한 단어의 모양이 쌍봉낙타의 등과 비슷하다는 뜻에서 이름붙었다.

올바른 예시
camelCase(**일반적인 변수 이름**)
isCamelCase(Boolean **타입의 변수 이름**)

문자열 연결

* **문자열과 변수/수식/함수 연결**
  1. **연결연산자** : ' '+**변수**+' '
  2. **백틱** : `<div> ${ **변수** } </div>`

* 함수모음

* 내장함수

1. isNaN(데이터) : 만약에 데이터가 숫자가 아니면 true 숫자이면 false

2. typeof(데이터) : 해당 데이터형 확인 [object , array , function 등등]

3. parseInt(데이터) : 해당 데이터를 정수형으로 변환

4. Math.random() : 0~1 사이의 실수 난수 생성
   - Math.random() : 0 ~ 1 사이의 실수 난수 발생
   - Math.random()+1 : 1 ~ 2 사이의 실수 난수 발생
   - Math.random()*45 : 0 ~ 45 사이의 실수 난수 발생
   - Math.random()*45+1 : 1~46 사이의 실수 난수 발생

5. setInterval() : 특정 시간마다 함수 실행
   - setInterval(함수명 , 시간(밀리초 = 1000/1초))
   - setInterval(()=>{ 실행문 } , 1000) : 1초마다 화살표함수 실행
   - clearInterval(변수명) : Interval 초기화
   - let 변수명 = setInterval(함수명 , 시간(밀리초 = 1000/1초))

6. new Date() : 현재 날짜/시간 반환 해주는 클래스
   - let 날짜 = new Date() : 현재 날짜/시간 를 변수에 저장