오픈소스SW실습
프론트엔드 로드맵

AI컴퓨터공학부 임현기


한 권으로 보는 프런트엔드 로드맵과 학습 가이드
Explore the Front-End Roadmap
김기수 지음
아는 만큼
보이는
프런트엔드
개발
FRONT
BACK
프런트엔드 로드맵 제시
현업에서 주로 쓰는 기술과 도구 소개
학습 순서와 방법 조언
길벗

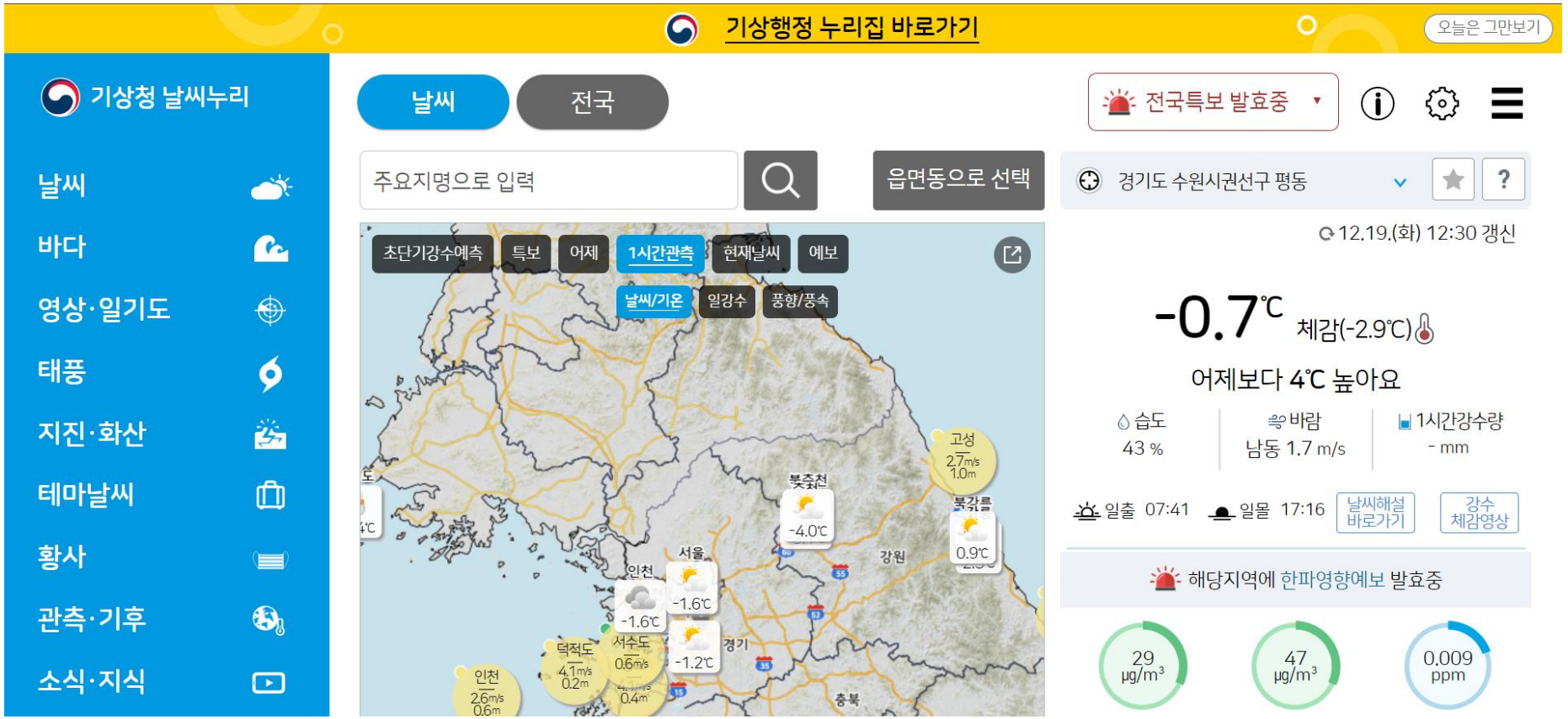
목차

1. 프론트엔드

1. 웹 개발의 구조
2. 프론트엔드의 등장
3. 프론트엔드 개발자가 하는 일

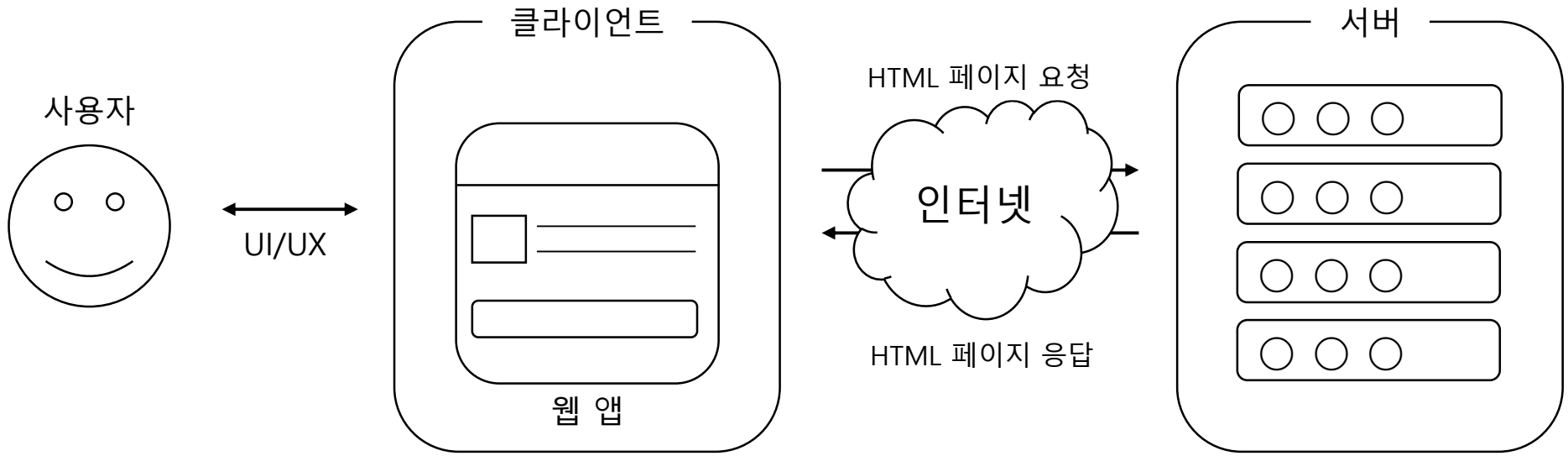
1.1. 웹 개발의 구조

- 웹 애플리케이션
  - 웹 브라우저에서 동작하며 사용자와 상호작용이 가능한 소프트웨어



1.1. 웹 개발의 구조

- 웹 개발
  - 웹 애플리케이션을 만드는 일



- 웹의 동작 방식
  - 사용자가 웹 브라우저를 통해 웹 애플리케이션에 작업 요청
  - 이를 서버가 받아 처리 한 뒤 결과 반환

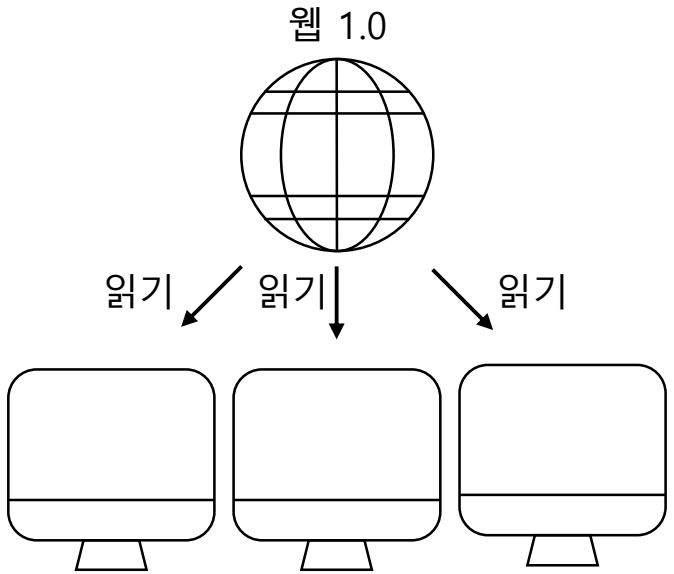
1.1. 웹 개발의 구조

- 클라이언트
  - 서버에 작업을 요청하는 컴퓨터 또는 응용 프로그램
  - 웹 브라우저가 클라이언트
  - 웹 애플리케이션은 웹 브라우저에서 실행되는 소프트웨어
- 서버
  - 클라이언트의 요청을 받아 처리하는 컴퓨터 또는 응용 프로그램

- 프론트엔드 개발
  - 사용자가 웹 앱을 사용할 수 있도록 눈에 보이는 화면과 기능 개발
- 백엔드 개발
  - 서버에서 동작하는 웹 앱, 데이터를 관리하고 처리하는 부분 개발

1.2. 프론트엔드의 등장

- 웹 1.0
  - 1990년부터 2004년
  - 이 시대의 웹: 정적
  - 사용자는 개발자가 만든 웹 페이지를 단순히 읽기만 함
  - 야후, 구글, 네이버, 다음

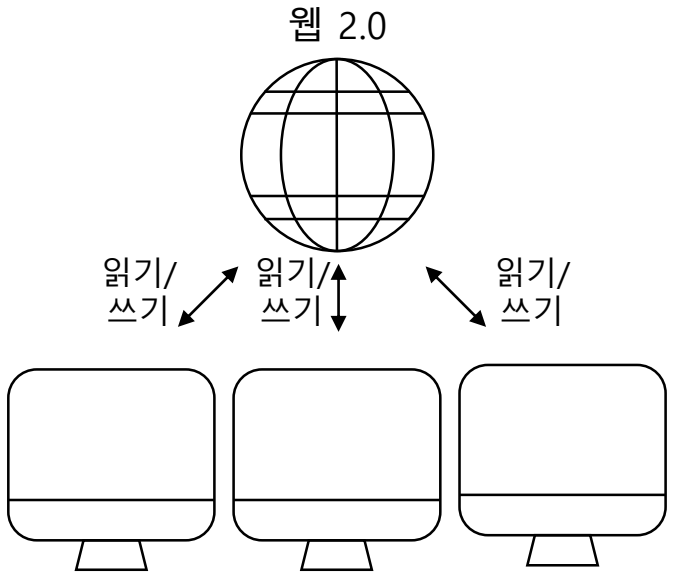


1.2. 프론트엔드의 등장

- 웹 1.0
  - 새로운 프로그래밍 언어 개발: CSS, 루비, PHP, 자바스크립트
  - 웹 디자이너 (페이지를 예쁘게 꾸미기)
  - 기술 난이도 낮음

1.2. 프론트엔드의 등장

- 웹 2.0
  - 2004년부터 현재
  - 이 시대의 웹: 상호작용 (interactive)
  - 정보를 읽기만 하는 것이 아니라 직적 작성/수정
  - 싸이월드, 페이스북, 트위터, 인스타그램 등



1.2. 프론트엔드의 등장

- 웹 2.0
  - 어도비(Adobe)에서 개발한 플래시(Flash)가 주력 기술이 됨
    - 애니메이션효과, 비디오 재생 가능
    - 웹 디자이너/개발자는 플래시를 할 줄 알아야함
  - 2010년 아이폰에서 플래시를 지원하지 않기도 하면서 급격한 변화를 맞게 됨
- 웹 3.0
  - 지능형 웹 (semantic web)
    - 웹 페이지에 담긴 내용을 이해하고, 개개인에 맞춘 정보를 제공
    - 대표적 기술: 인공지능, 블록체인, 사물 인터넷

1.2. 프론트엔드의 등장

- 이미지 태그의 등장
  - 1991년 HTML 태그 공개 (팀 버너스리)
    - , <menu>, <dir>,
  - 1993년 이미지 태그 <img> 등장
    - 시각적으로 풍성한 정보 제공

1.2. 프론트엔드의 등장

- CSS의 등장
  - 1994년 CSS 제안 (유럽입자물리연구소)
    - 웹 페이지에 시각적 디자인을 입히는 데 사용하는 언어
    - 웹 페이지 개발 업무 분리
      - 웹 페이지 구성 요소 배치
      - 전체적인 구조를 만드는 일
      - 구성 요소에 시각적 디자인 입히는 일
  - 1996년 CSS 정식 출시, 1998년 W3C(웹 표준 조직)의 권장 사항
  - CSS가 독립적 언어이지만 CSS가 적용된 웹 페이지가 단조로웠기 때문에 웹 개발에 고급 인력 필요하지 않음

1.2. 프론트엔드의 등장

- 자바스크립트의 등장과 웹 2.0 시대로의 전환
  - 1995년 자바스크립트 개발
    - 사용자와 웹 페이지가 서로 상호작용하며 동작할 수 있게 함
    - 웹 페이지를 만드는 데 보다 전문적인 기술자가 필요하게 되는 중요한 계기

    - 이 때부터 웹 2.0 시작으로 보지는 않고, 소셜 미디어가 등장한 2004년으로 봄

1.2. 프론트엔드의 등장

- AJAX의 등장 (Asynchronous JavaScript and XML, 에이잭스)
  - 비동기로 화면을 동적으로 구성할 수 있게 해주는 프로그래밍 기법
  - 비동기: 특정 작업이 다른 작업과 독립적으로 실행되는 방식

  - 이전에는 웹 페이지에 변경된 내용이 있으면 새로 고침을 해야 했음
    - 서버로부터 정보를 새로 받아오려면 웹 페이지 전체를 다시 읽어와야 했음

  - 구글의 비동기 기법
    - 2004년 지메일, 2005년 구글 지도

  - AJAX는 자바스크립트를 기반으로 대중적으로 사용
    - 이 때부터 전문적인 웹 개발자의 존재 부각

1.2. 프론트엔드의 등장

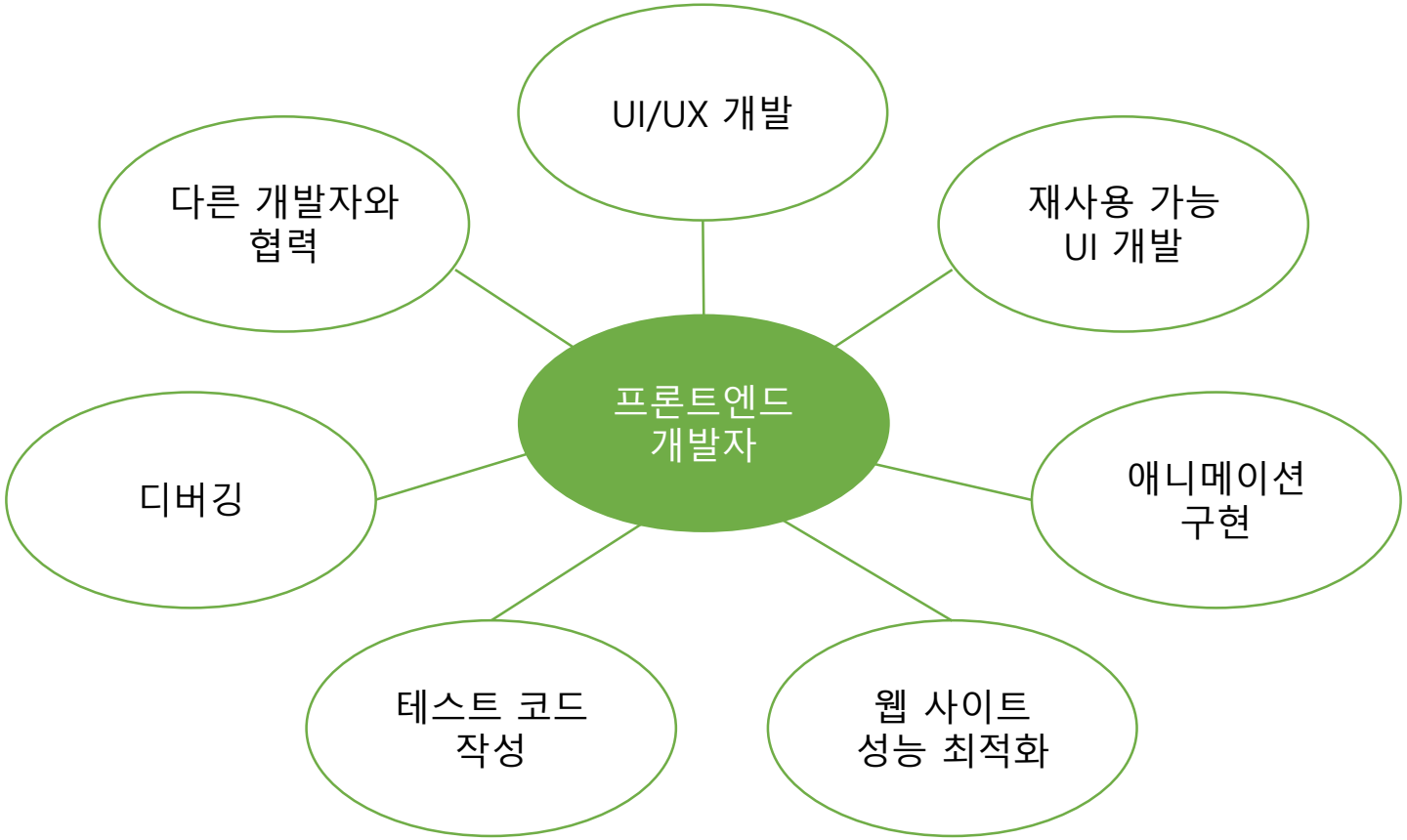
- 플래시의 몰락
  - 자바스크립트와 AJAX의 등장에도 플래시가 굳건
  - 매년 새로운 버전으로 업그레이드
    - 다양한 움직임, 동영상, 음악 같은 멀티미디어 요소 재생
    - 인터랙티브한 효과를 구현하는 핵심 기술

  - 그러나 2010년 기점으로 몰락의 길
    - 취약한 보안: 플래시가 악성 SW를 퍼트리는 등 보안상 위험
    - 성능 저하: 웹 페이지에서 자원을 많이 소모, 모바일 기기에서 배터리 문제
    - 접근성 문제: 플래시 포함된 HTML 구조를 해석/낭독 불가능 (시각 장애인)
    - 색인 생성의 어려움: 검색 엔진에서 플래시 예외 처리
    - HTML5와 AJAX의 등장: 플래시 없이 비디오/오디오 기능 지원 가능

1.2. 프론트엔드의 등장

- 웹 대용량화, 프론트엔드와 백엔드의 분리
  - 웹 2.0 시대 + AJAX 등장으로 웹이 더욱 발전
    - 사용자가 많아지고 트래픽도 몰림
    - 이에 따라 더 많은 정보 처리가 가능한 구조와 시스템이 필요
  - 웹 페이지를 효율적으로 제작하기 위해 프론트엔드/백엔드 분리
    - 프론트엔드: 처리한 데이터를 이용해 사용자 화면 구성
    - 백엔드: 서버 내부에서 데이터 처리

1.3. 프론트엔드 개발자가 하는 일



1.3. 프론트엔드 개발자가 하는 일

- 재사용 가능한 UI 개발
  - 앵귤러, 리액트, 뷰 등의 자바스크립트 프레임워크를 가지고 재사용할 수 있는 UI를 만듦
  - 이 UI는 향후 프로젝트를 진행하면서 UI의 일관성을 유지할 수 있고, 프로젝트 전반의 생상성 향상에 도움이 됨

- 웹사이트 성능 최적화
  - 로딩 속도, 반응 속도, 안정성 등의 요소를 개선해 사용자 경험 향상
  - 최적화에 사용되는 기술: 캐싱, 압축, 이미지 최적화 등

1.3. 프론트엔드 개발자가 하는 일

- 테스트 코드 작성
  - 작성한 코드의 품질 유지를 위해 테스트 코드 작성
  - 테스트 코드를 잘 만들면 다양한 브라우저와 기기에서 안정적인 서비스를 제공할 수 있음

- 다른 개발자와 협업
  - 코드, 문서, 데이터 등의 자원을 공유해 개발 속도 높임
  - 버전 관리 도구(Git, SVN)을 사용해 소스 코드 관리
  - 이슈 트래커(Jira)와 같은 도구 사용해 체계적으로 업무 할당/관리

2. 네트워크와 인터넷

1. 인터넷의 탄생과 발전 과정
2. 도메인와 DNS

2.1. 인터넷의 탄생과 발전 과정

- 일괄 처리 시스템
  - 1957년 이전, 한번에 하나의 작업만 처리
  - 점차 컴퓨터의 성능은 좋아지는데, 이용하는 사람이 한 명이라 성능을 최대한으로 활용하기 어려움

- 시분할 시스템
  - 시간을 쪼개 여러 사람이 컴퓨터 한 대를 사용
  - 매우 짧은 시간에 하나의 처리를 완료하고 다음 처리를 함
    - 마치 여러 개를 처리하는 것으로 보임
  - 컴퓨터 자원을 낭비하지 않고 효율적으로 사용

  - 단점: 갑자기 고장나거나 전력 차단시 모든 작업이 일시에 중단

2.1. 인터넷의 탄생과 발전 과정

- 컴퓨터 네트워크
  - 최초의 컴퓨터 네트워크 아파넷(ARPANET)
    - 미국 군사 조직인 고등연구계획국
    - 연구 및 기술 개선을 거쳐 1983년 TCP/IP를 적용한 완전체 출범
    - 오늘날 인터넷의 원형으로 평가

2.1. 인터넷의 탄생과 발전 과정

- 인터넷의 발전
  - 월드 와이드 웹 (1989-1995년)
    - 1989년 팀 버너스가 하이퍼텍스트(다른 텍스트의 대한 링크가 포함된 텍스트) 기반의 문서 공유 시스템 제안
    - 이후 이 아이디어를 발전시켜 월드 와이드 웹 구축
    - 인터넷의 폭발적인 성장
  - 닷컴 버블 (1995-2000년)
    - 많은 인터넷 서비스 회사(닷컴으로 불림)가 설립
  - 닷컴 몰락(2000-2003년)
    - 가장 큰 문제: 인터넷 속도 - 서비스나 사업하기 어려움
    - 연구 및 개발 지속했지만 경제적인 문제 극복하지 못하고 줄줄이 파산
    - 하지만 많은 투자와 개발로 인해 인터넷 발전에 큰 도움

2.1. 인터넷의 탄생과 발전 과정

- 인터넷의 발전
  - 닷컴 몰락 후 회복 (2003-2007년)
    - 닷컴 몰락 후 극복한 회사들이 비즈니스 모델을 구축
    - 구글, 아마존, 이베이 같은 기업이 강력한 시장 위치를 확보
    - 인터넷 환경을 발전시킬 수 있는 검색엔진 최적화, 클라우드 컴퓨팅 등 기술 등장

  - 모바일 인터넷 (2007년-현재)
    - 1993년 IBM의 Simon이라는 스마트폰 출시, 대중화에 실패
    - 2007년 애플의 아이폰 공개

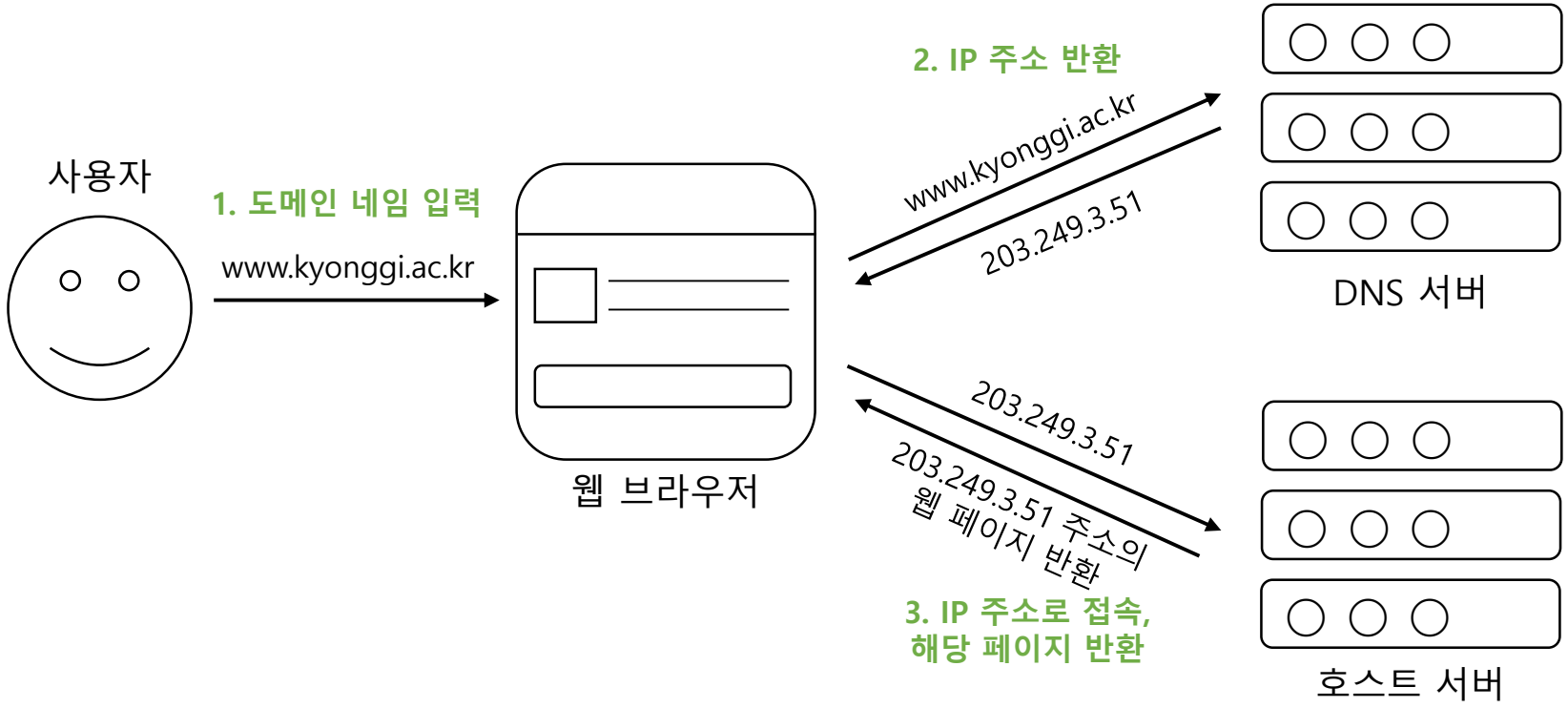
2.2. 도메인과 DNS

- IP(Internet Protocol) 주소
  - 컴퓨터 네트워크에서 컴퓨터의 위치를 식별할 수 있는 주소
  - 초기 아파넷에서는 IP 주소를 hosts.txt라는 파일에 저장, 수동 관리
    - 민간에 공강된 후 폭발적으로 확장함에 따라 파일로 관리 불가

- 도메인 네임
  - IP 주소를 표현하는 고유한 숫자를 나타내기 어렵기 때문에 주소와 1:1로 매칭되며 기억하기 쉬운 도메인 네임을 사용

2.2. 도메인과 DNS

- DNS (Domain Name System)
  - 도메인 네임으로 IP 주소가 문자열로 대체되었지만, 어딘가에서 관리할 필요가 있음 → DNS



3. HTML, CSS, 자바스크립트

1. HTML
2. CSS
3. 자바스크립트

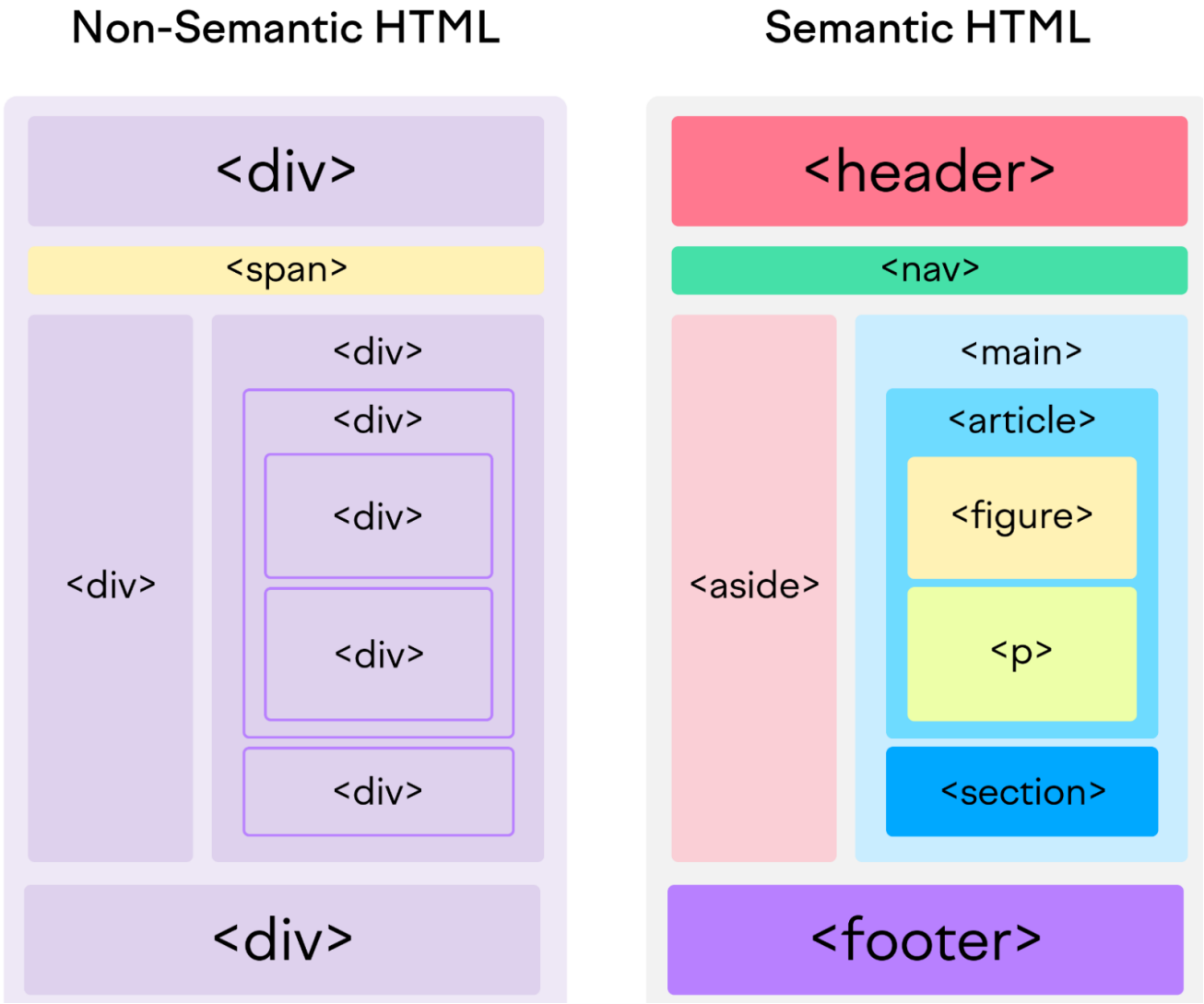
3.1. HTML

- 프론트엔드 개발
  - 웹 브라우저에 시각적으로 렌더링되는 UI를 개발하는 것
    - 렌더링: 서버로부터 소스 코드를 읽어와 웹 브라우저에서 보이는 그래픽 형태로 출력하는 과정
  - HTML (Hypertext Markup Languel)
    - 렌더링된 UI를 만드는 데 사용하는 언어

3.1. HTML

- 태그
  - 일반 태그
    - <시작 태그>콘텐츠</종료 태그>
  - 초창기에는 태그 자체에 의미를 부여하지 않고, 시각적인 효과만 표현
    - 예: 는 글자를 굵게, 는 이탤릭체
  - 시맨틱 태그
    - HTML이 발전하면서 웹 페이지의 효과적인 유지/보수 및 코드 확장성을 위해 콘텐츠와 시각적인 효과를 분리하는 방향이 됨
    - 콘텐츠를 보다 의미있게 전달 + 문서의 구조를 효율적으로 구분
      → 의미론적 태그의 필요성 부각 → HTML5의 시맨틱 태그
    - 사람이 이해하기 쉽도록 이름만 보고 역할이나 위치를 알 수 있게 함

3.1. HTML


Non-Semantic HTML

Semantic HTML
<header>
<nav>
<aside>
<main>
<article>
<figure>
<p>
<section>
<footer>

3.1. HTML

- 속성
  - 태그만으로 모든 정보를 나타내기 어려움
    - 예: <a> 태그는 하이퍼텍스트를 위한 태그로, 다른 콘텐츠를 연결
    - 그래서 HTML은 태그와 더불어 속성을 사용
    - 링크제목

- 학습 방법
  - 태그 암기하려 하지 말기
  - 태그를 정확히 알고 사용하기
  - 다양한 예제 코드 분석하기
  - 공식 문저와 온라인 자료 참고

3.2. CSS

- 주요 특징
  - 캐스케이딩(cascading)
    - 우선순위 판별
      - 중요도: CSS 속성 마지막에 !important 키워드를 붙이면 가장 높은 우선순위
        - !important 키워드가 여러 개 사용됐다면 작성 순서가 더 늦은 것이 우선 적용
      - 명시성: 선택자가 얼마나 구체적인지를 나타내는 값
        - 내부적으로 명시성 값이 다음과 같이 정의 (총합이 높은 스타일이 우선 적용)
          - 인라인 선택자: 1000
          - 아이디 선택자: 100
          - 클래스/가상클래스/속성 선택자: 10
          - 요소/가상 요소 선택자: 1
      - 작성 순서
        - 늦게 작성된 속성일수록 우선순위가 높음

3.2. CSS

- 주요 특징
  - 상속
    - 부모 요소에 적용된 속성이 자식 요소에 자동으로 적용되는 현상
    - 예: <body> 태그 내에 <h1> 태그를 사용해 제목 작성했을 때,
      <html> 태그에 CSS로 적용하면 순차적으로 <html>,<body>,<h1>에 상속

- 적용 방법
  - 내부 스타일 시트
  - 외부 스타일 시트

3.2. CSS

- 적용 방법
  - 내부 스타일 시트

```
<head>
        <style>
                /* CSS 코드 */
        <style>
</head>
```

  - 외부 스타일 시트

```
<head>
        <link rel="stylesheet" href="style.css">
</head>
```

3.2. CSS

- 적용 방법
  - 인라인 스타일

```
<body>
        <p style="color: red"> sample text</p>
</body>
```

- 기본 문법
  - 선택자 + 선언부

```
h1{ color: red; }
```

    - h1: 선택자
    - {color: red;}: 선언부
      - color: 속성, red: 값

3.2. CSS

- 학습 방법
  - 선택자 지정 방법 정확히 알기
    - flukeout.github.io
  - 속성과 값 연습하기
  - 스타일 호환성 이해하기
    - 같은 CSS 속성과 값이라도 웹 브라우저의 종류에 따라 다르게 보일 수 있음
  - 반응형 디자인과 미디어 쿼리 학습하기
    - 미디어 쿼리: 웹 페이지를 렌더링하는 장치에 따라 다른 스타일 적용
      - 데스크톱과 스마트폰 화면 크기에 따라 다른 스타일 처리 가능
  - 공식 문서와 자료 참고

3.3. 자바스크립트

- 자바스크립트
  - 웹 페이지에 복잡한 기능 구현
  - 자바스크립트가 없다면 일정하고 동일한 데이터만 보여주는 정적 웹 페이지가 될 것

  - 예: 로그인 페이지
    - 아이디/비멀번호 입력 후 로그인하면 유효성 검증
    - 검증 후 그에 맞는 화면 보여주기

3.3. 자바스크립트

- 자바스크립트
  - HTML: 웹 구조 설계
  - CSS: 웹 페이지 디자인
  - JS: 웹 동작 구현

  - 스크립트 언어
    - 기존의 소프트웨어를 제어하는 용도로 쓰이는 언어
    - 문법이 단순하고 쉽고, 컴파일 없이 바로 실행
    - 속도가 느림

3.3. 자바스크립트

- 적용 방법
  - 내부 스크립트

```
<body>
        <script>
                // 자바스크립트 코드
        </script>
</body>
```

  - 외부 스크립트

```
<body>
        <script src="script.js"></script>
</body>
```

3.3. 자바스크립트

- 적용 방법
  - 인라인

```
<body>
        <button onclick="자바스크립트 코드"></button>
</body>
```

3.3. 자바스크립트

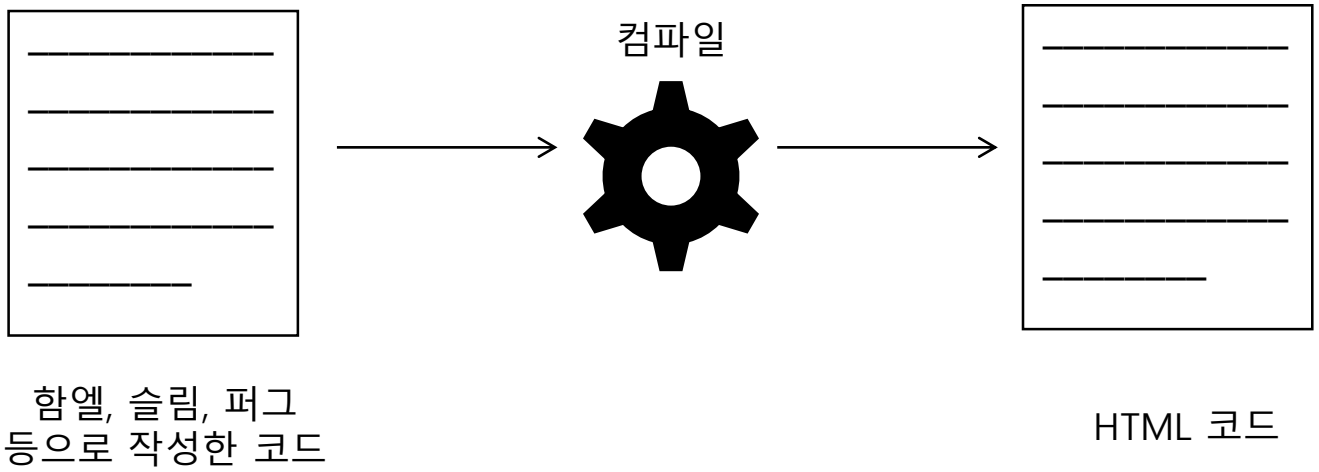
- 기본 문법
  - 실행문
  - 선언 및 할당
  - 자료형
    - 기본 자료형: 문자, 숫자, 논리, Undefined, Null, Symbol 등
    - 참조 자료형: 함수, 배열, 객체

4. HTML, CSS, 자바스크립트 심화 기술

1. HTML 전처리기
2. CSS 전처리기
3. CSS 후처리기
4. CSS 방법론
5. 타입스크립트

4.1. HTML 전처리기

- HTML 전처리기
  - 기존의 HTML 문법을 확장/개선하여 작성한 코드를 HTML 코드로 변환하는 도구
  - 함엘, 슬림, 퍼그 등



4.1. HTML 전처리기

- 함엘
  - Haml, HTML abstraction markup language
  - 2006년 처음 공개
  - 기존의 HTML 문법보다 더 간결하고 가독성 높은 문법 제공
    - 문서 구조를 들여쓰기로 표현
    - 태그의 중첩 표현 간소화
    - 태그와 속성을 짧은 키워드로 표현

4.1. HTML 전처리기

- 함엘

함엘

```
!!! 5
%html
   %head
      %meta(charset="utf-8")
      %title Haml To HTML
   %body
      %h1 haml
      %h2 html
```

HTML

```
<!DOCTYPE html>
<html>
   <head>
      <meta charset="utf-8">
      <title>Haml to HTML</title>
   </head>
   <body>
      <h1>haml</h1>
      <h2>html</h2>
   </body>
</html>
```

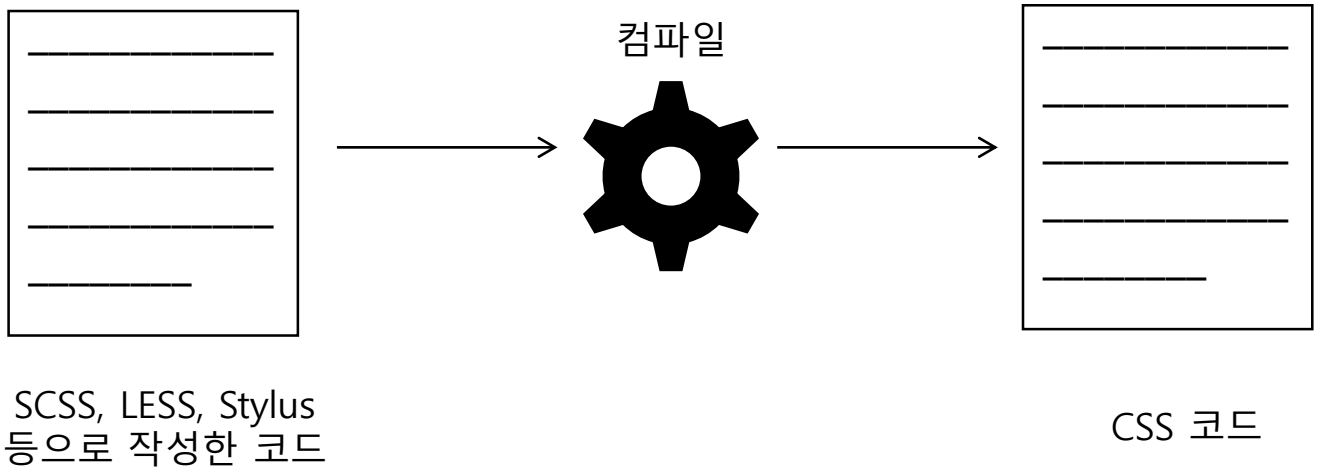
4.1. HTML 전처리기

- 슬림
  - 깃허브에서 여러 개발자들이 오픈 소스 프로젝트로 개발
  - HTML의 핵심 구문을 줄이는 것을 목표로 만들어짐

- 퍼그
  - 2011년 공개
  - 함엘의 영향을 많이 받았고, 자바스크립트와의 호환성도 좋음 (함엘과 슬림은 루비 언어 기반)

4.2. CSS 전처리기

- CSS 전처리기
  - 기존의 CSS 문법을 확장/개선하여 작성한 코드를 CSS 코드로 변환하는 도구
  - SCSS, LESS, Stylus 등



4.2. CSS 전처리기

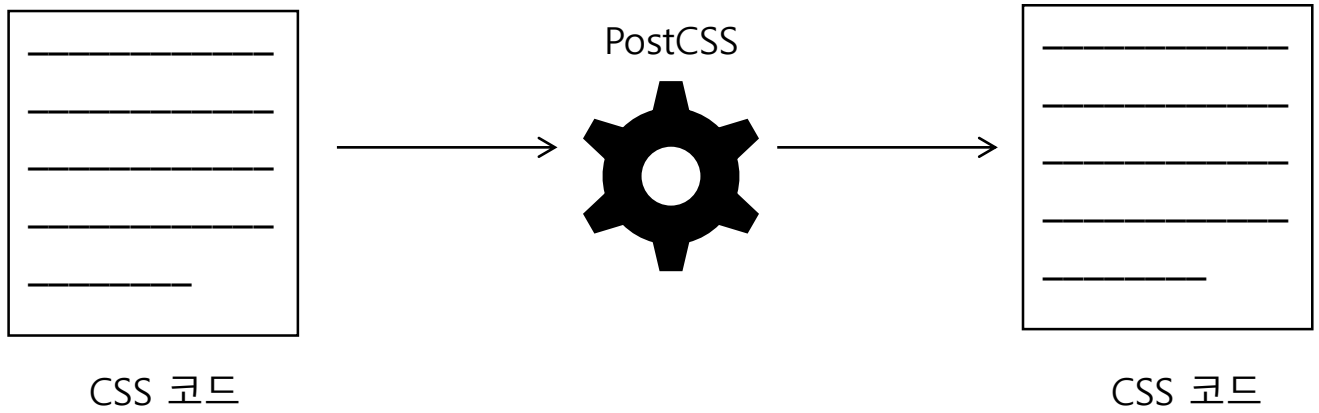
- SCSS
  - SASS(Syntactically Awesome Style Sheets, 2006년)로 개발된 후 새롭게 개편하여 SCSS라는 이름으로 공개
  - 변수, 가져오기, 중첩, 확장, 재사용, 연산, 조건문, 반복문 등 프로그래밍 언어에 있을 기능을 제공
    - 새로운 문법이 추가되어 배우기 어렵다는 단점이 있음

- LESS
  - 2009년 공개, SCSS보다 더 간결하고 편리한 기능 추가

  - 현업에서 SCSS, LESS가 주로 사용, 사용 빈도 높음

4.3. CSS 후처리기

- CSS 후처리기
  - 완전하게 작성된 CSS 문법을 가지고 자바스크립트 플러그인을 이용해 CSS 문법 스타일을 변환하는 도구
  - 전처리기와 달리 기존의 CSS 문법을 그대로 적용 가능
  - 예: PostCSS



4.4 CSS 방법론

- CSS 전처리기/후처리기
  - 결국 CSS라는 언어를 문법적으로 뜯어 고쳐 개선하려는 것
  - 일각에서 CSS를 문법적으로 고치지 않고, 좀 더 효율적으로 작성해 불편을 해결하려고 함 → CSS 방법론

- CSS 방법론
  - CSS 코드를 구조화하고 효율적으로 관리하기 위한 일련의 관행/규칙
  - 코드의 복잡성과 유지/보수의 어려움 등 문제점을 극복하려고 함

4.4 CSS 방법론

- OOCSS(Object Oriented CSS)
  - 객체 지향 디자인 원칙을 CSS에 적용해 CSS를 객체 지향 디자인이 적용된 프로그래밍 언어처럼 관리하기 쉽게 만들고자 함

  - 예: 구조와 외형의 분리

Delete

```
btn {
    width: 86px;
    height: 39px;
    padding: 9px 17px 10px 17px;
    background-color: green;
    border: none;
    border-radius: 5px;
    font-size: 16px;
}
<button class="btn">Delete</button>
```

```
.btn {
    width: 86px;
    height: 39px;
    padding: 9px 17px 10px 17px;
}
.delete {
    background-color: green;
    border: none;
    border-radius: 5px;
    font-size: 16px;
}
<button class="btn delete">Delete</button>
```

4.4 CSS 방법론

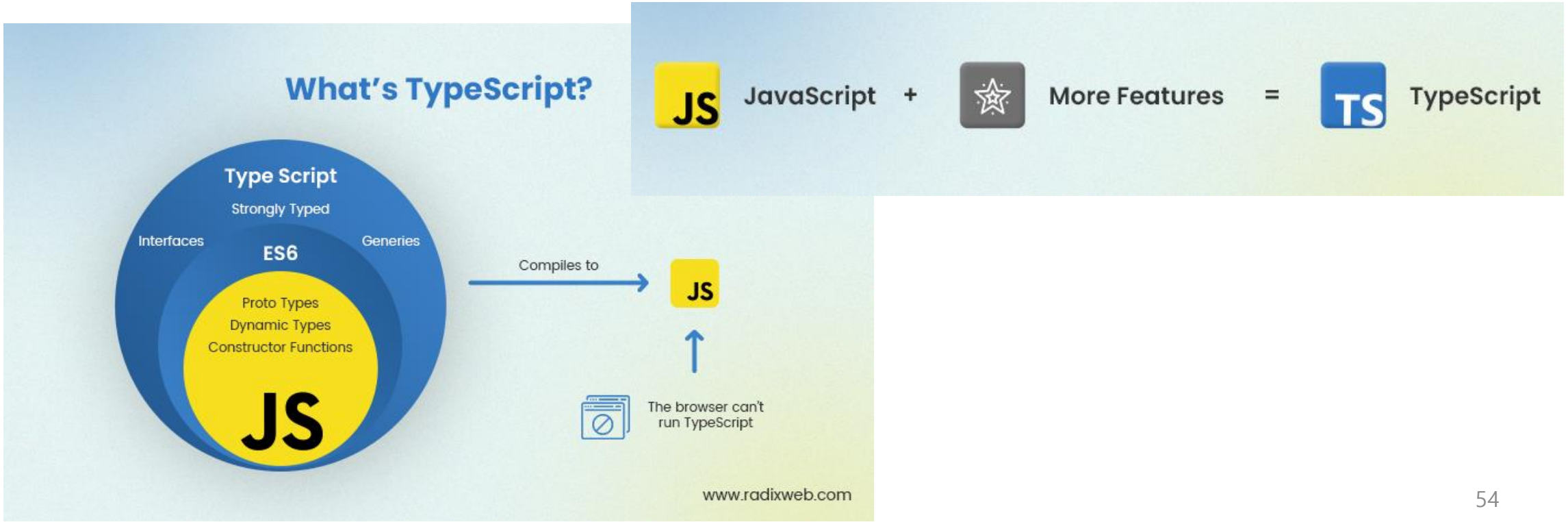
- SMACSS(Scalable and Modular Architecture for CSS)
  - 확장 가능한, 모듈러 방식의 아키텍쳐를 지원하는 방법론
  - 다섯 가지 규칙(기본, 레이아웃, 모듈, 상태, 테마)를 지켜야 함

- BEM
  - 스타일을 정의하기 위한 class 속성의 명명 규칙에 초점을 둠
  - class 속성은 CSS를 적용하기 위한 식별자로, HTML의 모든 태그는 class 속성을 사용할 수 있음

  - 방법론을 자체적으로 정의해 사용하는 경우들이 있으므로 많은 시간을 할애하여 공부하는 것은 권장되지 않음

4.5. 타입스크립트

- HTML, CSS 전처리기 등이 등장한 것처럼 자바스크립트도 전처리기가 등장하여 대체하고 있음
  - 라이브스크립트(LiveScript), 커피스크립트(CoffeeScript), 바벨(Babel), 타입스크립트(TypeScript) 등이 있으며 이중 타입스크립트가 가장 많이 사용됨



4.5. 타입스크립트

- 타입스크립트
  - 마이크로소프트에서 2012년에 공개, 자바스크립트 확장 언어
  - 자바스크립트보다 더 엄격하게 정의한 자료형 사용

  - 선택적 정적 타입 검사
    - 자바스크립트는 데이터가 동적으로 할당돼 런타임 시 해당 변수에 값이 할당될 때까지 변수의 자료형을 알지 못함

4.5. 타입스크립트

const numbers = ‘hello’;	자바스크립트
const numbers: number = 10;	타입스크립트
const numbers: number = ‘hello’;	타입스크립트 → 오류 발생
const numbers: any = “hello”;	타입스크립트

4.5. 타입스크립트

- 타입스크립트
  - 인터페이스
    - 객체와 같은 참조자료형의 정적 타입 검사를 쉽게 할 수 있음

```
interface People {
    name: string;
    age: number;
    gender: string;
}
```

타입스크립트
인터페이스 작성

```
interface People {
    name: string;
    age: number;
    gender: string;
}
const student: People = {
    name: "철수",
    age: 20,
    gender: "M",
};
```

인터페이스를 활용한
객체 선언

4.5. 타입스크립트

- 타입스크립트
  - 자바스크립트를 먼저 공부하는 것을 권장
  - 타입스크립트는 학습 진입 장벽이 있고, 개발 생산성이 떨어진다는 단점이 있음
    - 안정성을 위한 추가적인 코드 작업이 필요

5. 개발 도구

1. 소스 코드 에디터
2. 버전 관리 시스템
3. 코드 포매터
4. 린터
5. 패키지 매니저
6. 모듈 번들러

5.1. 소스 코드 에디터

- 브라켓
  - 2014년, 어도비 에지에서 사용할 수 있는 웹 개발용 소스 코드 에디터

- VSCode
  - 마이크로소프트에서 개발한 무료 소스 코드 편집기
    - 강력한 인텔리센스 지원
    - 디버깅 기능 내장
    - 깃 명령 탑재
    - 확장 및 사용자 정의 가능

5.2. 버전 관리 시스템

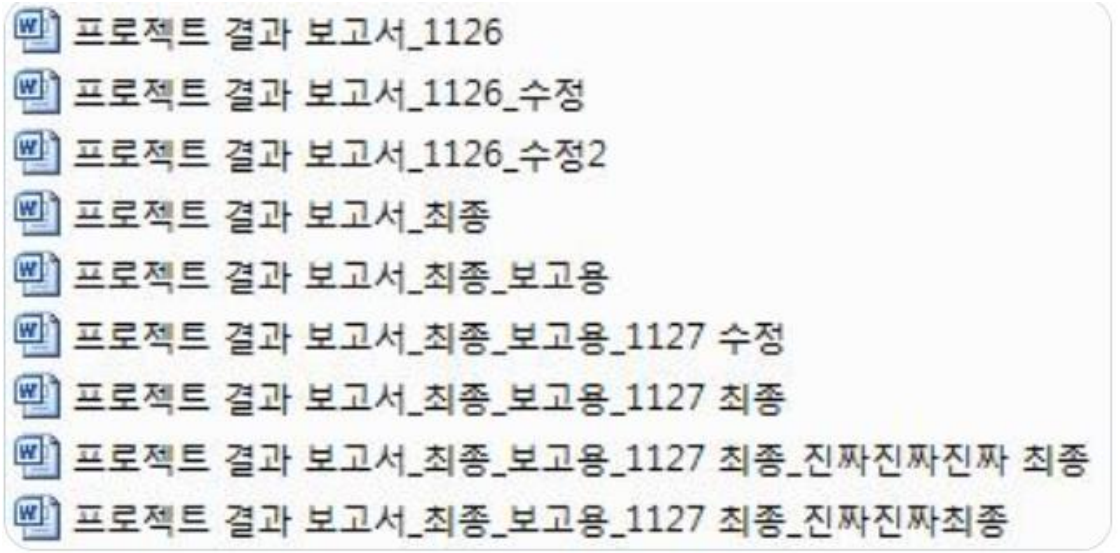
- 버전 관리 시스템(Version Control System, VCS)
  - 코드나 파일을 시간에 따라 기록/추적/관리 하는 시스템

  - 주요 기능
    - 기록: 코드의 변경 사항을 하나의 버전으로 관리해 기록하는 기능,
      누가, 언제, 어떤 내용으로 코드를 변경했는지 정보가 남음
    - 추적: 기록된 각 버전을 확인하는 기능
    - 분기: 하나의 코드를 여러 사람이 동시에 작업할 수 있도록 코드의 특정 시점으로 분기하는 기능, 코드의 특정 시점을 기억하는 세이브 포인트를 협업하는 사람들에게 하나씩 나눠주고, 그 시점을 기준으로 작업함
    - 병합: 분기된 코드를 하나로 합치는 기능
    - 백업 및 복구: 현재 작성 중인 코드를 임시로 저장하고, 코드를 기록했던 과거의 특정 시점으로 돌아가는 기능, 코드에 치명적인 결함이 발견돼 급히 수정해야 할 때 기능 사용

5.2. 버전 관리 시스템

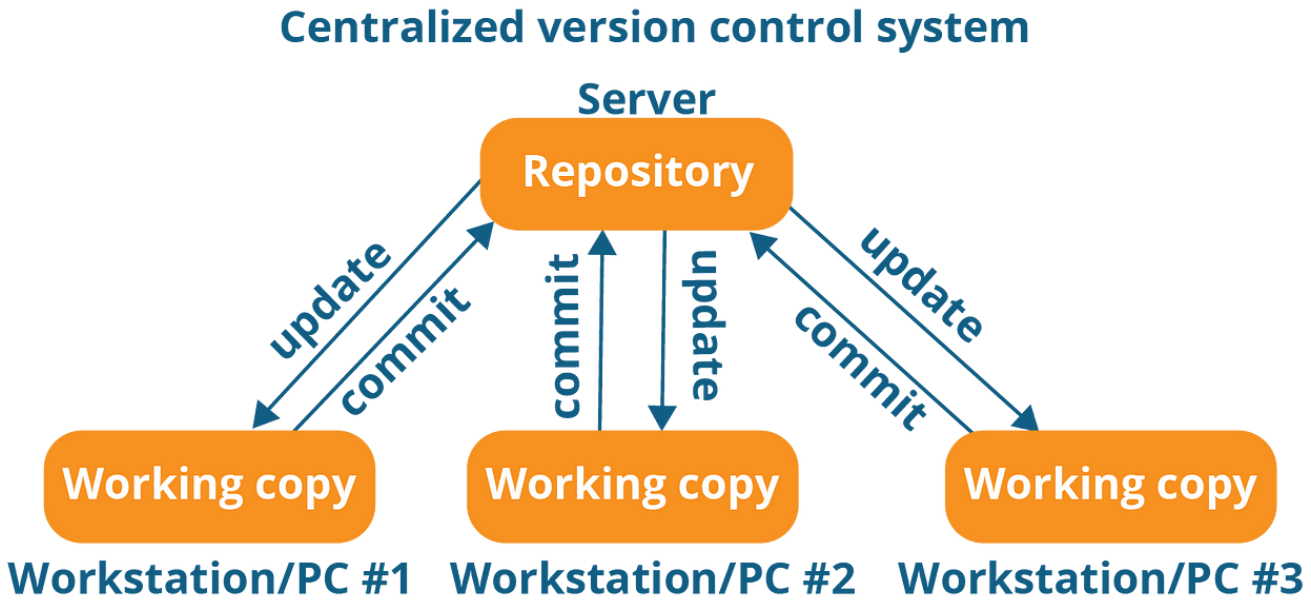
- 버전 관리 시스템의 종류
  - 세 가지 방식: 로컬, 중앙 집중식, 분산

  - 로컬 버전 관리 시스템 (Local VCS)
    - 가장 간단한 방법, 작업 중인 폴더의 파일을 다른 폴더로 복사
    - 대표적인 소프트웨어: RCS(Revision Control System)



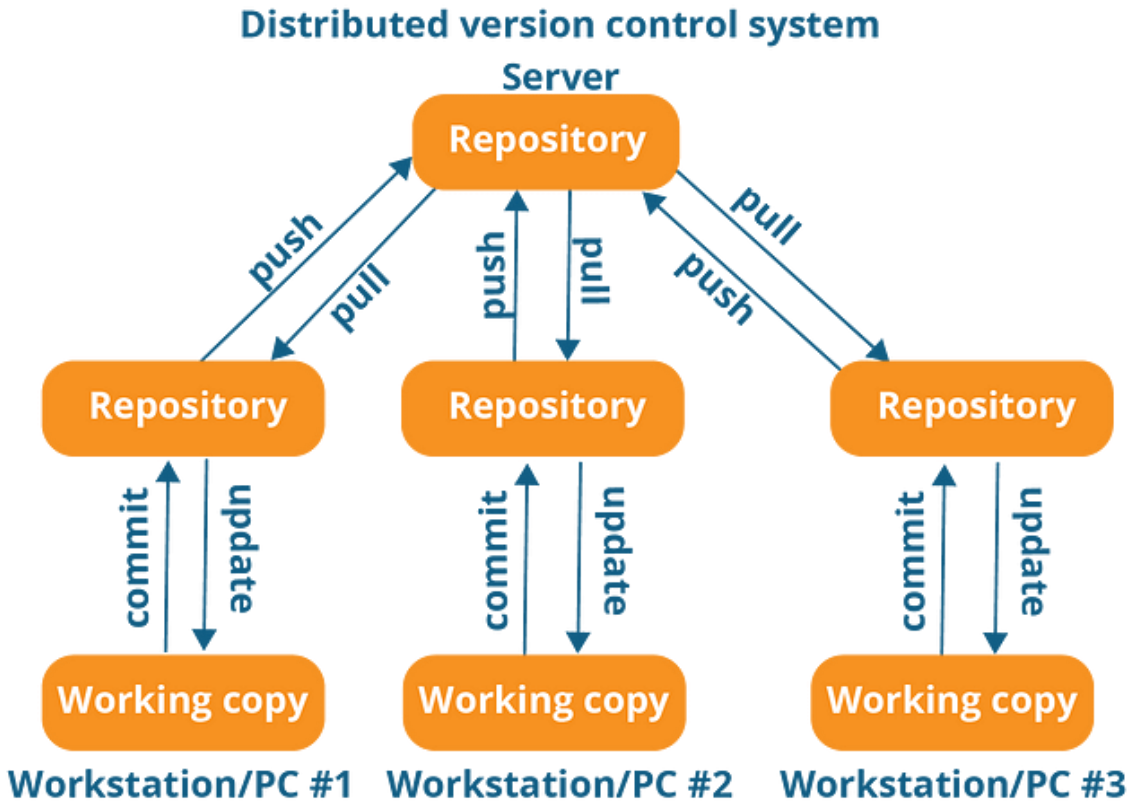
5.2. 버전 관리 시스템

- 중앙 집중식 버전 관리 시스템 (Centeral VCS)
  - LCVS는 DB를 각 사용자가 가지고 있어 여러 사람과 협업하기 불편함
  - DB를 별도의 서버에 설치하고, 각 파일의 변경 사항을 서버에 기록하는 시스템
  - CVC(Concurrent Versions System), SVN(Subversion)
  - 단점: DB가 설치된 서버에 문제가 생기면 사용자 모두 영향 받음



5.2. 버전 관리 시스템

- 분산 버전 관리 시스템(Distributed VCS)



5.2. 버전 관리 시스템

- 분산 버전 관리 시스템(Distributed VCS)
  - 깃(Git): 2005년 개발되어 지금까지 쓰이는 대표적 소프트웨어

- 웹 기반 버전 관리 저장소
  - 깃허브: 2008년
    - 저장소 관리 페이지 제공
    - 호스팅 서비스 제공
    - 협업 및 커뮤니티 제공
  - 깃랩: 2013년
    - 기본적으로 깃허브 기능들 제공
    - 지속적 통합, 지속적 배포 기능 제공 (CI/CD 파이프라인 기능 제공)
      - 빌드 → 테스트 → 배포 프로세스 자동화
    - 세부적 권한 설정 가능
  - 비트버킷

5.3. 코드 포매터

- 등장 배경
  - 개발자 저마다의 코드 스타일을 가짐
  - 여러 명이 협업할 때 서로의 스타일 파악에 부담
  - 일부 회사에서는 '코드 스타일 가이드' 제공 → 또 다른 스트레스

  - 코드 포매터: 도구로 코드 스타일을 일관되게 변경하는 것

```
// 스타일 1
if(true) {};
// 스타일 2
if(true) {
}
// 스타일 3
if(true)
{
}
```

5.3. 코드 포매터

- 프리티어(Prettier)
  - 국내외 가장 인기 있는 코드 포매터
  - VSCode와 호환성이 좋음
    - 유사한 확장 프로그램 많음 → 제작자 Prettier 확인 후 설치

5.4. 린터

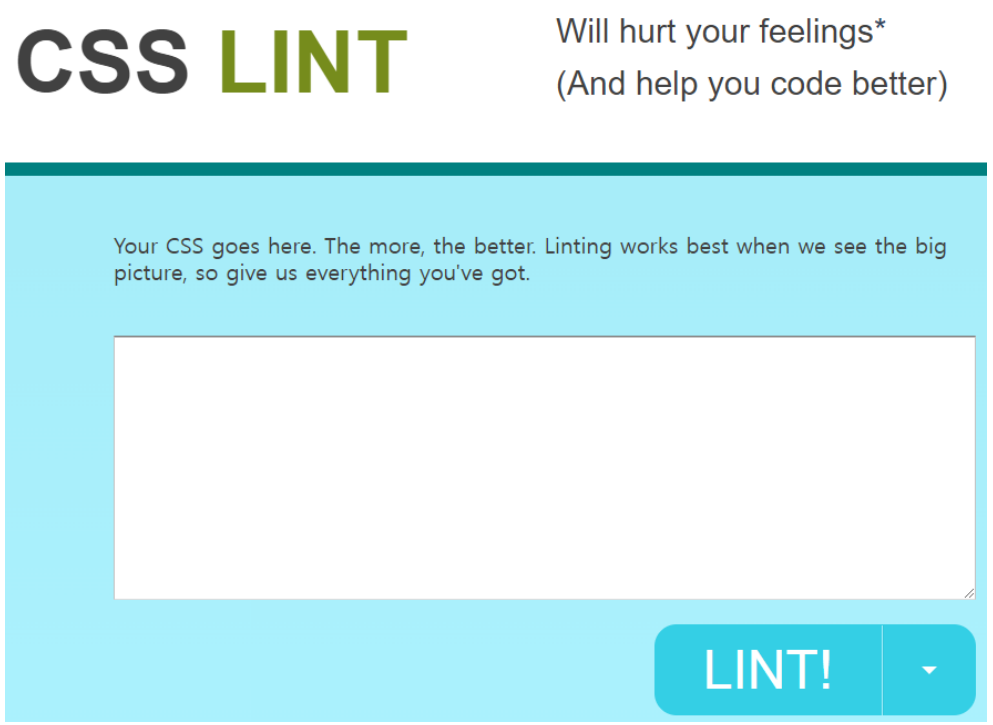
- 린터
  - 작성한 코드를 정적으로 분석해 문법적으로 오류가 발생할 만한 곳을 사전에 검사하고 올바른 코드를 작성할 수 있도록 도와주는 도구
  - '정적으로 분석' - 코드를 실행하지 않고 분석

- 등장 배경
  - 코드의 양이 많아지면 어느 부분에서 오류가 발생할지 예측 어려움
  - 협업 시 상호간 영향을 주는 코드가 많아짐
  - 내가 작성하지 않은 부분에서 발생한 오류를 바로 잡기 위해 다른 사람이 작성한 코드를 임의로 수정할 일이 생기기도 함 → 협업 어려움

```
const name = '철수';
// 몇 백 줄의 코드 생략
const name = '영희';
// const 키워드 중복 불가, 하지만 협업시 보장되지 않음
// 린터로 검사하면 'Parsing error: Identifier 'name' has already been declared'라는 경고 메시지 확인
```

5.4. 린터

- HTML+CSS 린터
  - W3C의 마크업 검증 서비스 (정식 린터는 아님)
  - W3C의 CSS 검증 서비스 (정식 린터는 아님)
  - CSS 린트
  - 스타일린트 (CSS) - VSCode에서 확장 기능 사용 가능



5.4. 린터

- 자바스크립트+타입스크립트 린터
  - JS린트(2002년)
    - 별도 설치 없이 사용 가능 (www.jslint.com)
  - JS힌트(2011년)
    - 별도 설치 없이 사용 가능 (www.jshint.com)
  - ES린트(2013년)
    - 자바스크립트뿐 아니라 타입스크립트 코드까지 린팅
    - 별도 설치 없이 사용 가능 (eslint.org/play)

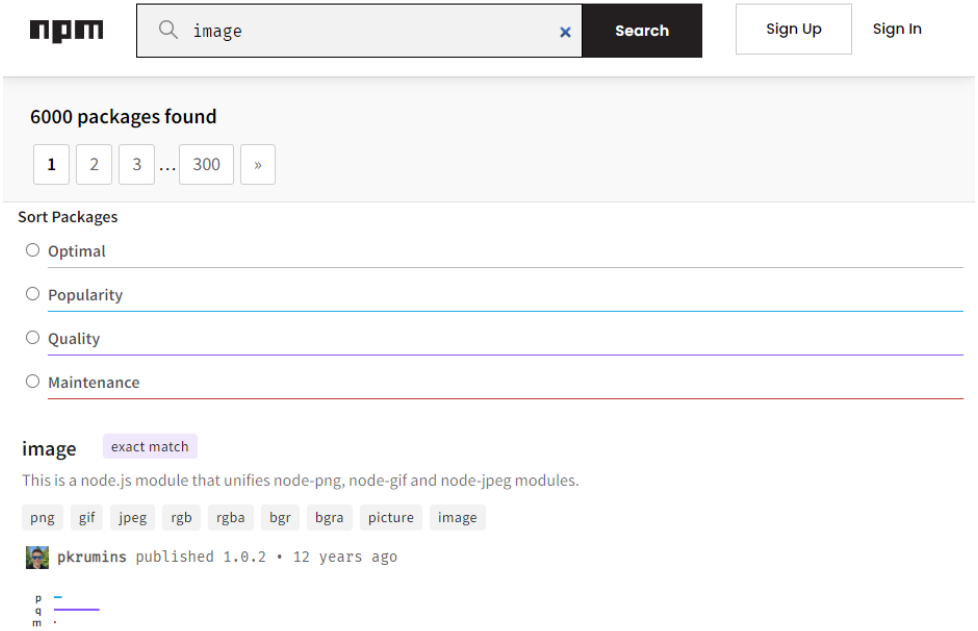
5.5. 패키지 매니저

- 패키지 매니저
  - 패키지를 관리하는 작업을 자동화하고 안전하게 처리하기 위한 도구
  - 패키지: 기능을 구현하기 위해 작성한 라이브러리 또는 코드 집합

- npm
  - 과거 프론트엔드 개발은 모든 것을 개발자가 직접 작성/구축하는 것
  - 오늘날은 '최소한의 코딩'
  - 변화에 가장 큰 영향을 준 것이 2010년 npm(node package manager)

5.5. 패키지 매니저

- npm
  - Node.js 기반으로 작성된 패키지를 관리하기 위한 온라인 저장소와 커맨드 라인 도구(CLI)를 제공하는 개발 도구
    - 라이브러리르 모아놓은 온라인 저장소
  - 오픈소스 프로젝트
    - 누구나 자유롭게 npm을 통해 패키지를 올리거나 내려받을 수 있음



5.6. 모듈 번들러

- 모듈 번들러
  - 자바스크립트 파일 여러 개를 웹 브라우저에서 실행할 수 있게 하나의 파일로 묶는 데 사용하는 도구
  - 자바스크립트나 타입스크립트를 위한 개발 도구 (HTML, CSS X)

5.6. 모듈 번들러

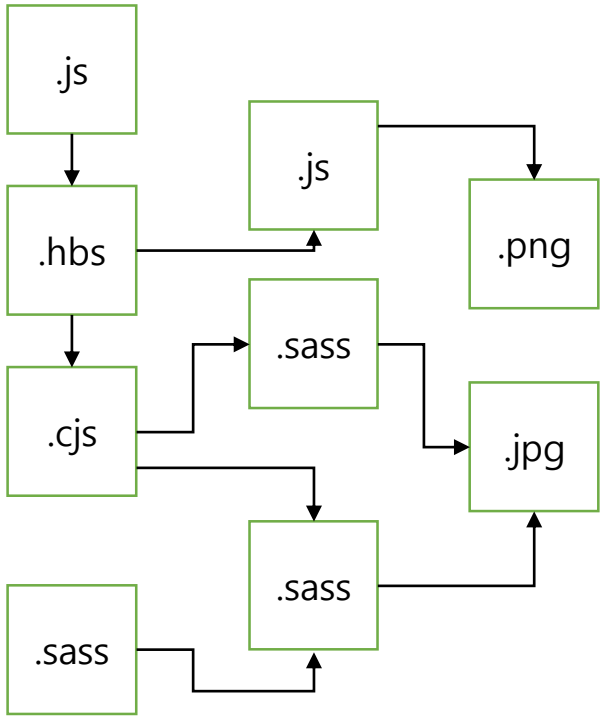
- 등장 배경
  - 코드 분할화
    - 자바스크립트 파일을 기능 또는 단위별로 분할
  - 웹 성능 향상
    - 코드 분할이 성능 향상을 보장하지 않음
    - '파일 용량' 뿐만 아니라 파일을 내려받는 '요청 횟수'도 중요한 요인
      - 10kb 1개 vs 1kb 5개
    - 따라서 파일을 여러 개로 나눠 작성했더라도 최종적으로 웹 서비스로 배포할 때 하나로 합치는 것이 더 좋음
  - 종속성 문제
    - 여러 개의 코드를 하나로 합친다 하더라도 의존성 문제를 해결하지 않으면 웹 서비스가 정상적으로 작동하지 않을 수 있음

5.6. 모듈 번들러

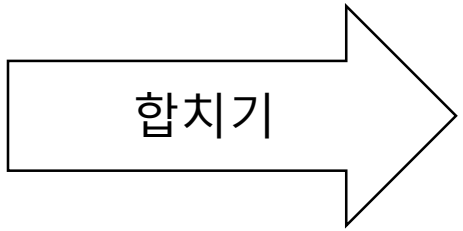
- 웹팩
  - 대표적 모듈 번들러
    - 브라우저리파이(Browserify), 이에스빌드(Esbuild), 파슬(Parcel), 롤업(Rollup), 웹팩
    - 이 중 웹팩(Webpack)이 가장 인기
  - 자바스크립트 뿐 아니라 HTML, CSS, 각종 이미지 파일 등 하나의 웹 서비스를 구성하는 모든 파일 관리

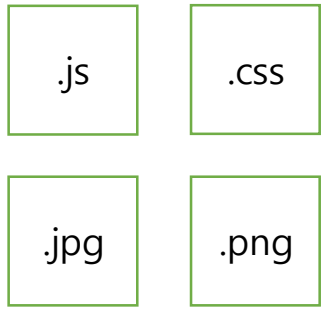
5.6. 모듈 번들러

- 웹팩



종속성이 있는 모듈





정적인 자산

5.6. 모듈 번들러

- 웹팩 구성요소
  - 엔트리
    - 프로젝트에서 사용하는 웹 자원을 변환하기 위한 최초의 진입점
  - 아웃풋
    - 웹팩이 모듈 번들링을 끝내고 최종적으로 산출되는 파일을 내보내는 경로
  - 로더
    - 자바스크립트, HTML, CSS, 이미지, 폰트 등을 하나로 묶기
  - 플러그인
    - 로더가 완료할 수 없는 추가적인 작업을 함
    - 로더를 통해 나온 결과물 최적화, 또는 형태를 바꾸는 일
  - 모드
    - 배포 용도, 개발 용도 인지를 구분

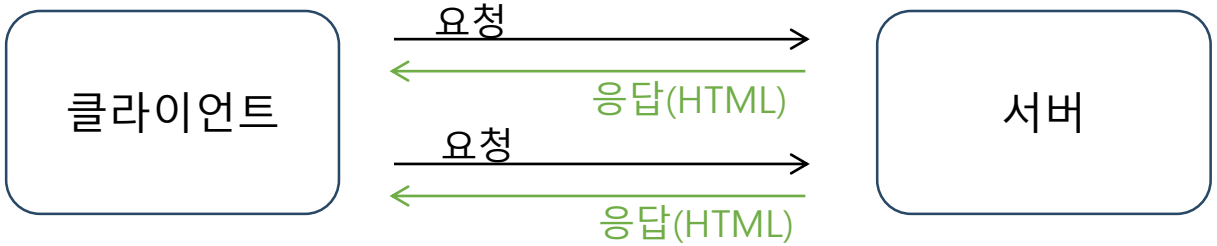
6. 디자인 패턴과 프레임워크

1. 웹 애플리케이션 디자인 패턴
2. 자바스크립트 프레임워크
3. CSS 프레임워크

6.1. 웹 앱 디자인 패턴

- 디자인 패턴
  - 웹 앱을 개발할 때 일반적으로 사용되는 접근 방식 또는 모범 사례
  - 개발시 디자인 패턴을 적용하면 시간이 지나도 유지/보수가 편리, 확장성 좋은 웹 사이트를 만들 수 있음
  - MPA, SPA, SSR이 있음

- MPA (Multi Page Application)
  - 웹 페이지에서 서버로 데이터를 요청하고 응답을 받을 때 매번 새로운 HTML 페이지를 받는 방식
  - 웹 탄생했을 때부터 지금까지 사용됨



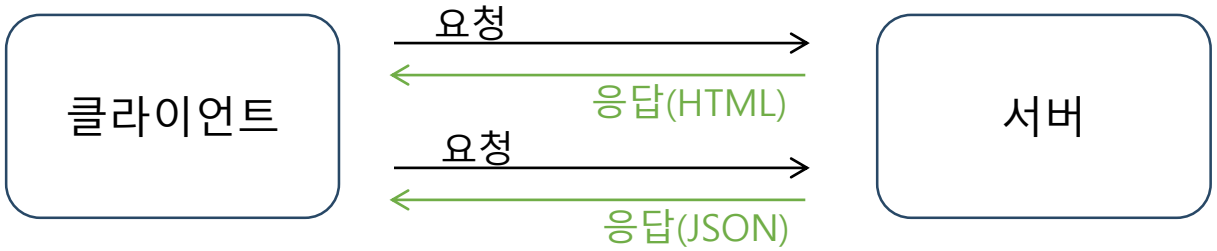
6.1. 웹 앱 디자인 패턴

- MPA (Multi Page Application)
  - 장점
    - 모든 페이지가 분리되어 있어 검색 엔진 최적화에 용이
    - 구글 애널리틱스와 같은 웹 앱 분석 솔루션과 통합하기 쉬움
  - 단점
    - 응답으로 항상 새 HTML 페이지를 받기 때문에 갱신하기 위해 항상 새로고침 해야함 → 속도 저하, 성능 문제
    - 모든 페이지가 분리되어 개발해야 할 양이 많아짐
    - 많은 페이지 양만큼 보안, 유지, 보수 어려움
  - AJAX가 등장하기 전까지 모든 웹에서 사용
    - 현재도 이베이, 아마존 등에서 사용 중

6.1. 웹 앱 디자인 패턴

- SPA (Single Page Application)
  - 웹 페이지에서 처음 응답받을 때 딱 한번 HTML, CSS, 자바스크립트 같은 자원을 내려받고, 다음 요청부터는 응답받은 데이터로 필요한 부분만 변경하는 방식
    - 응답 데이터는 XML, CSV, HTML 등 가능한데, JSON(JS Objection Notation)이 가장 많이 사용

  - 응답받은 데이터를 웹 페이지에서 직접 변경
    - CSR(Client Slide Rendering) 이라고 하기도 함



6.1. 웹 앱 디자인 패턴

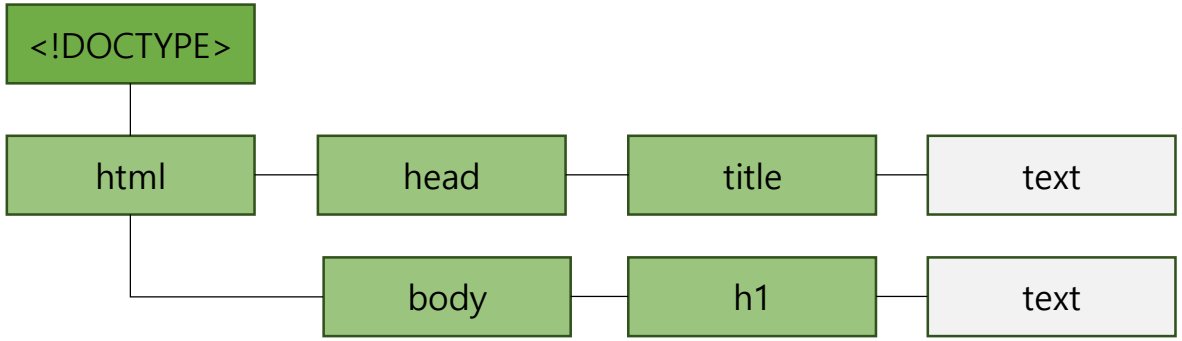
- SPA (Single Page Application)
  - 장점
    - 다시 로딩하지 않고 변경되는 부분만 갱신 → 페이지 갱신에 따른 부담이 적음
    - 새로고침이 발생하지 않아 사용자 경험(UX)가 좋아짐
  - 단점
    - 모든 페이지가 분리되어 있지 않아 검색 엔진 최적화에 매우 불리
    - 하나의 웹 페이지 내에서 데이터가 변경되기 때문에 보안 측면에서 취약
    - 웹 브라우저의 히스토리를 따로 관리하지 않으므로 URL 개념이 없음

    - 이런 단점에도 불구하고 오늘날 가장 중요한 디자인 패턴
    - 앵귤러, 리액트, 뷰 모두 SPA 디자인 패턴을 기반으로 함

6.1. 웹 앱 디자인 패턴

- SPA (Single Page Application)
  - DOM (Document Object Model)
    - 웹 브라우저에서 표시되는 HTML, CSS 요소를 자바스크립트가 이해할 수 있도록 객체화해 제공하는 모델
    - 웹 브라우저가 화면에 표시되는 구성 요소에 대해 DOM을 만들어 제공하면 자바스크립트는 DOM을 이용해 웹 브라우저에 표시되는 구성 요소를 제어

```
<!DOCTYPE html>
<html lang="ko">
   <head>
      <title>DOM tree structure</title>
   </head>
   <body>
      <h1>DOM tree structure</h1>
   </body>
</html>
```



6.1. 웹 앱 디자인 패턴

- SPA (Single Page Application)
  - DOM (Document Object Model)
    - DOM 생성이 끝나면 웹 브라우저는 자바스크립트가 DOM에 접근할 수 있도록 브라우저 API, DOM API와 같은 접근 방법을 제공
    - 자바스크립트는 DOM을 조작해 변경이 필요한 부분을 갱신
    - 응답받은 데이터로 필요한 부분만 변경 → DOM에 접근해야 한다는 것을 의미

    - 자바스크립트가 DOM에 직접 접근하면 내부적으로 연속적인 처리가 발생, 속도가 성능 면에서 매우 불리 (DOM 트리가 복잡할수록 더 큰 문제)

    - SPA 디자인 패턴을 기반으로 하는 자바스크립트 프레임워크는 이 문제를 해결하기 위해 DOM을 제어하는 알고리즘을 제공
      - 앵귤러: 증가 DOM 방식의 알고리즘
      - 리액트, 뷰: 가상 DOM 방식의 알고리즘

6.1. 웹 앱 디자인 패턴

- SSR (Server Side Rendering)
  - 검색 엔진 로봇
    - 웹 사이트를 수집
    - 인터넷을 통해 콘텐츠를 다운로드하고 인덱싱하는 역할
      → 사용자에게 올바른 정보를 제공
    - SPA 디자인 패턴은 웹 사이트에 표시되는 데이터가 클라이언트에서 업데이트
      → 검색 엔진 로봇은 서버에서 실행되어 가져갈 수 있는 정보가 없음
  - SSR 디자인 패턴은 이런 문제를 해결
    - 웹 페이지를 서버에서 렌더링
    - 장점
      - 초기 로딩 속도가 빠름 (서버에서 이미 렌더링한 데이터를 응답받음)
      - 검색 엔진 최적화에 유리

6.2. 자바스크립트 프레임워크

- SPA 디자인 패턴은 오늘날 프론트엔드 개발에 주로 사용
  - 화면 업데이트 요청을 AJAX로 하고,
  - 응답은 자바스크립트의 데이터 포맷 중 하나인 JSON으로 받아, 클라이언트에서 직접 처리

  - 자바스크립트를 이용해 웹 페이지를 새로 고침하지 않고, 서버에서 변경된 데이터만 받아 처리
    → 웹 사이트에서 자바스크립트로 처리해야할 범위가 많아짐

6.2. 자바스크립트 프레임워크

- 앵귤러
  - 구글의 앵귤러팀과 개인 및 기업이 주도하는 타입스크립트 기반 오픈 소스 프레임워크
  - 2009년 구글에서 사이드 프로젝트로 개발, 2016년 앵귤러2 공개
  - 앵귤러2는 자바스크립트 대신 타입스크립트를 사용
  - 현재 앵귤러16 버전까지 출시

6.2. 자바스크립트 프레임워크

- 앵귤러
  - 사용 이유
    - 구글의 전폭적인 지원
    - 타입스크립트 사용 (안정성)
    - 한국어 지원 (https://angular.kr/docs)
    - 뛰어난 브라우저 호환성
  - 사용하지 않는 이유
    - 높은 학습 곡선: 타입스크립트가 자바스크립트보다 학습 어려움
    - 개발이 복잡성: 컴포넌트 단위 관리
    - 상대적으로 느린 속도: 절대적인 속도가 느리지는 않음
    - 낮은 국내 인지도

6.2. 자바스크립트 프레임워크

- 앵귤러의 특징
  - 증가 DOM 사용
    - 새로운 DOM 트리를 생성하는 동안 기존의 DOM 트리를 따라 이동하면서 변경 사항을 파악
    - 변경 사항이 없으면 메모리 할당하지 않고, 있으면 그 부분을 업데이트하기 위한 최소한의 메모리만 할당
  - HTML, CSS 문법 사용
    - 별도 언어 사용할 필요 없이 HTML, CSS를 그대로 사용 가능
    - (리액트는 JSX, 뷰는 템플릿 문법을 사용)

6.2. 자바스크립트 프레임워크

- 리액트
  - 메타플랫폼스의 오픈 소스 프로젝트
    - 2011년 페이스북 개발자가 늘고 기능도 다양해짐에 따라 코드 관리가 어려워짐
    - 코드를 효율적으로 관리할 수 있는 근본적인 대안이 필요
      → FaxJs 프로토타입 프레임워크
      - 원활한 클라이언트-서버 렌더링
      - 상태 기반의 선언적 업데이트
      - 구성 요소의 손쉬운 재사용
      - 2012년 인스타그램 서비스에도 사용
    - 2013년 5월, 리액트 공개

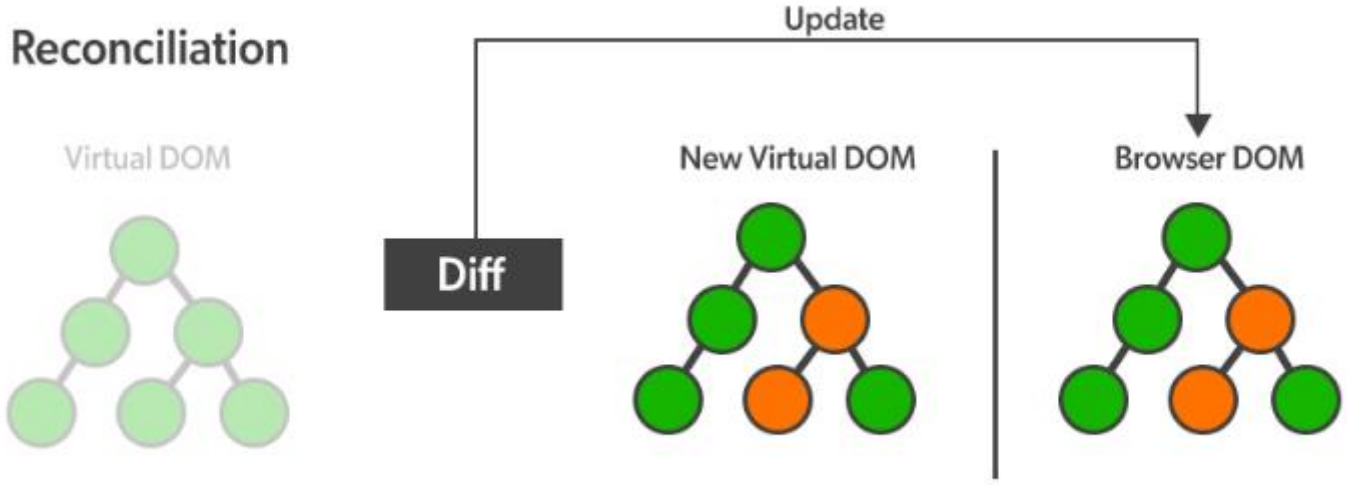
  - 라이브러리의 성격과 프레임워크 성격을 같이 가짐
    - 제한하는 규칙이 없고, 필요할 때 리액트의 기능을 가져다 사용할 수 있음
    - 컴포넌트 개념을 사용할 때는 프레임워크 성격

6.2. 자바스크립트 프레임워크

- 리액트
  - 사용 이유
    - 메타플랫폼스의 전폭적인 지원
    - 빠른 속도
  - 사용하지 않는 이유
    - 빠른 업데이트로 인한 학습 부담
    - 배워야 하는 많은 도구: 웹팩, 바벨, JSX 등
    - 제한적인 기능

6.2. 자바스크립트 프레임워크

- 리액트의 특징
  - 가상 DOM 사용
    - 웹 브라우저가 생성한 DOM을 그대로 복사한 가상의 DOM
    - 화면 구성 요소를 업데이트하려고 할 때, 가상 DOM을 기반으로 변경된 부분을 반영한 새로운 가상 DOM을 먼저 그림
    - 그리고 이전에 관리하던 가상 DOM과 새로 그린 가상 DOM을 비교해 다른 부분을 찾고, 실제 DOM에서 변경된 부분만 업데이트



6.2. 자바스크립트 프레임워크

- 리액트의 특징
  - 단방향 데이터 바인딩
    - 데이터 바인딩: 웹 브라우저에 보이는 데이터와 자바스크립트 객체에 저장된 데이터를 일치시키는 것
    - 단방향 바인딩: 자바스크립트 객체에 저장된 데이터가 웹 브라우저의 구성 요소로만 전달하는 것
      - 사용자가 웹 브라우저에서 데이터를 입력하더라도 리액트에서는 이 데이터를 자바스크립트 객체에 반영하지 않음 (코드의 일관성, 유지/보수 편함)

6.2. 자바스크립트 프레임워크

- 뷰
  - 자바스크립트 기반 오픈 소스 프로그레시브 프레임워크
    - 프로그레시브: 웹 생태계에 맞게 유연하고 점진적으로 적용해나간다
    - 늦게 출시되었지만 단순하다는 특징 덕분에 점유율이 빠르게 오르는 중
  - 리액트, 앵귤러에 장점들을 가져옴
    - 앵귤러: 양방향 데이터 바인딩, 템플릿 구문 등
    - 리액트: 가상 DOM, 구성 요소 기반 접근

6.2. 자바스크립트 프레임워크

- 뷰
  - 뷰를 사용하는 이유
    - 간단한 설치
    - 프레임워크 적용의 유연성
    - 낮은 학습 곡선 https://vuejs.org/guide/introduction.html
  - 뷰를 사용하지 않는 이유
    - 커뮤니티 활용의 한계 (창시자 에번 유가 중국인)
    - 뚜렷한 후원처 부재
    - 플러그인 부재

6.2. 자바스크립트 프레임워크

- 자바스크립트 프레임워크의 장점
  - 선언형 프로그래밍: 결과를 중요시하는 방식

```
const numArr = [1, 2, 3, 4, 5];
const arr = [];
const doubleNum = (num) => num * num;
for (let i=0; i<numArr.length; i++) {
   arr[i] = doubleNum(numArr[i]);
}
```

명령형

```
const numArr = [1, 2, 3, 4, 5];
const arr = numArr.map( (v) => v * v );
```

선언형

    - map() 메소드가 어떤 방식으로 수행되는지 알 수도 없고, 알 필요도 없음

  - 컴포넌트 기반
    - 컴포넌트는 재사용이 가능한 독립된 모듈
    - 코드의 유지/보수/재사용성 뛰어남

컴포넌트 기반으로 작성된 뷰코드

```
<template>
   <div>
      <Header />
      <Main />
      <Footer />
   </div>
</template>
```

6.3. CSS 프레임워크

- CSS 프레임워크
  - 웹 브라우저를 통해 사용자가 보는 UI는 비슷한 것이 많음
    - 매번 CSS를 새로 작성하는 것은 번거로운 일
    - 이를 개선하기 위해 CSS 프레임워크를 사용
  - CSS 코드 집합
    - 반복되는 UI 빠르게 구축
    - UX(사용자 경험) 균일화
  - 예: 부트스트랩, 테일윈드 CSS

6.3. CSS 프레임워크

- 부트스트랩
  - 반응형 및 모바일 친화적인 웹 사이트 개발에 사용, 오픈 소스
  - CSS 프레임워크 중 가장 높은 점유율

  - 사용하는 이유
    - 간단한 설치: npm, CDN 라이브러리 사용하면 쉽게 설치 가능
    - 반응형 그리드 지원
      - 미리 정의된 그리드 시스템을 제공, 화면 레이아웃 구성시 스타일을 따로 고민하지 않아도 손쉽게 그리드 레이아웃을 적용할 수 있음
      - https://getbootstrap.com/docs/5.3/layout/grid/

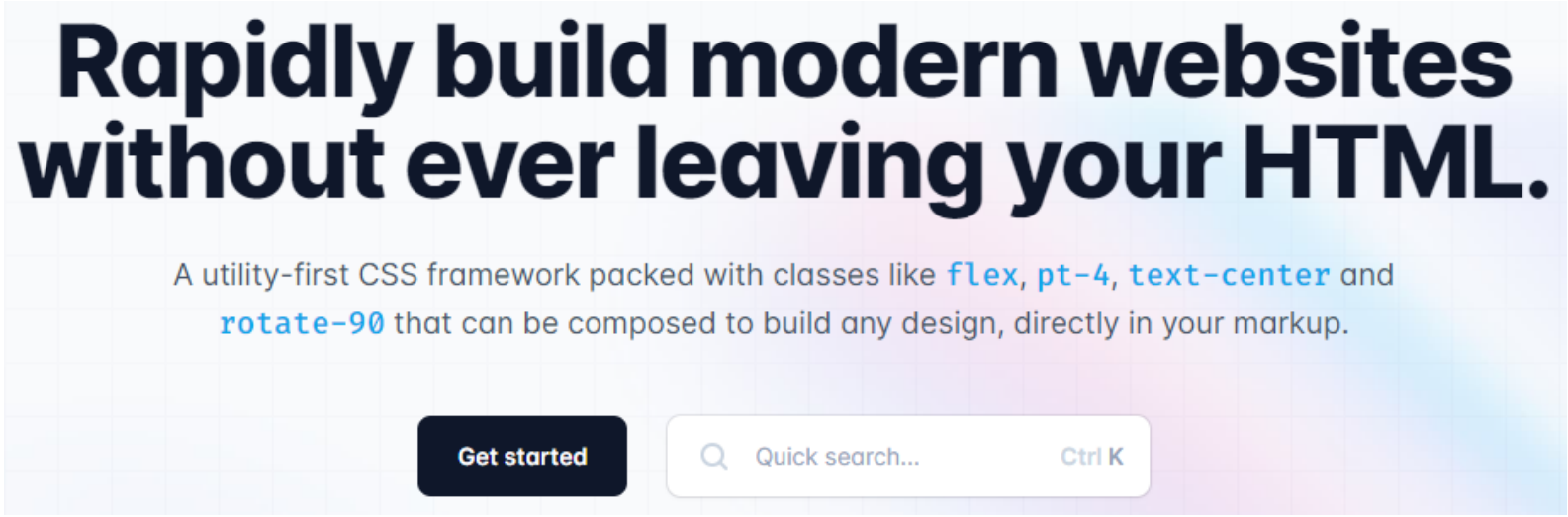
    - 브라우저 호환성 지원
    - 컴포넌트 제공
    - 프레임워크 친화적: 앵귤러, 리액트, 뷰에서 사용 가능한 별도의 플러그인 있음

6.3. CSS 프레임워크

- 부트스트랩
  - 사용하지 않는 이유
    - 일관된 스타일: 개성있는 웹 사이트를 만들 경우 적절치 않음
    - 느린 속도
    - 긴 학습 기간
      - 자체적으로 정의된 스타일, 규칙 등을 준수해 작성해야 함
      - 그리드 시스템을 이해하지 않고는 사용하기 어려움

6.3. CSS 프레임워크

- 테일윈드 CSS
  - 테일윈드연구소에서 개발한 CSS 프레임워크



6.3. CSS 프레임워크

- 테일윈드 CSS
  - 사용하는 이유
    - 유틸리티 퍼스트 컨셉트
      - 디자인을 위해 일련의 완성된 클래스명을 사용하지 않고, 단일 기능의 CSS 속성을 통해 클래스명을 혼합해 사용하는 방식

Sign in

```
<style>
   .btn{
      background-color: green;
      font-weight: 700;
      padding-left: 1rem;
      padding-right: 1rem;
      ...
   }
</style>
<button class="btn">Sign in</button>
```

```
<button class="bg-green-500 text-white font-bold py-2 px4 rounded>
   Sign in
</button>
```

6.3. CSS 프레임워크

- 테일윈드 CSS
  - 사용하는 이유
    - 브라우저별 기본 스타일 초기화

```
<button>Sign in</button>
```

      - 크롬, 파이어폭스, 사파리마다 모두 다르게 버튼이 나타남
      - 리셋 또는 노말라이즈: 기본 스타일 시트를 모두 지워버리는 작업

6.3. CSS 프레임워크

- 테일윈드 CSS
  - 사용하지 않는 이유
    - 긴 적응 기간
      - 유틸리티 클래스들을 새로 다 알아야 함
    - 큰 파일 용량
      - 유틸리티 퍼스트 콘셉트의 프레임워크이므로, 일관된 스타일을 한번에 제공하는 부트스트랩에 비해 용량이 큼
      - 코드 빌드 시 불필요한 클래스는 삭제해주는 플러그인 사용 필요
    - 재사용의 어려움
      - 예를 들어 20개의 버튼을 만든 후 수정을 한다면?

```
<button class="bg-green-500 text-white font-bold py-2 px4 rounded>
  Sign in
</button>
<button class="bg-green-500 text-white font-bold py-2 px4 rounded>
  Sign in
</button>
...
```