

# Intro to Web Scraping

# Goal

- 웹 크롤링과 스크래핑의 차이를 인지한다.
- 웹 스크래핑을 위해 알아야 할 웹에 대한 기초지식을 이해한다.
- requests와 BeautifulSoup으로 정적인 사이트 스크래핑을 수행할 수 있다.
- Selenium을 활용하여 동적인 사이트 스크래핑을 수행할 수 있다.
- Ajax 혹은 API 기반의 원본데이터 스크래핑을 수행할 수 있다.
- 스크래핑에 대한 스케줄링 방법을 인지한다.
- 웹 스크래핑 프로젝트를 github flow로 수행할 수 있다.

# Web Crawling

# Web Crawling

- 조직적 자동화된 방법으로 월드 와이드 웹을 탐색하는 것
- Scraping: 데이터를 수집하는 행위
- Parsing: 문장 혹은 문서를 구성 성분으로 분해하고 위계관계를 분석하여 문장의 구조를 결정하는 것

# Web Crawling 작업의 진행단계

- 목표 설정 및 진행 가능성 검토
- 시나리오 시뮬레이션
- 솔루션 구현
- 솔루션 시행 및 사후평가

# Web Crawling 시 주의해야 할 사항

- 관련 법령을 위반하지 않아야 함(저작권법, 컨텐츠산업진흥법, 개인정보보호법 등)
- 웹사이트 이용약관을 준수해야 함
- 타인의 경제적 이익을 침해하지 않아야 함(영리적 목적의 스크래핑 금지)
- 개인정보를 수집하지 않아야 함
- 웹사이트 서버에 물리적 타격을 입히지 않아야 함

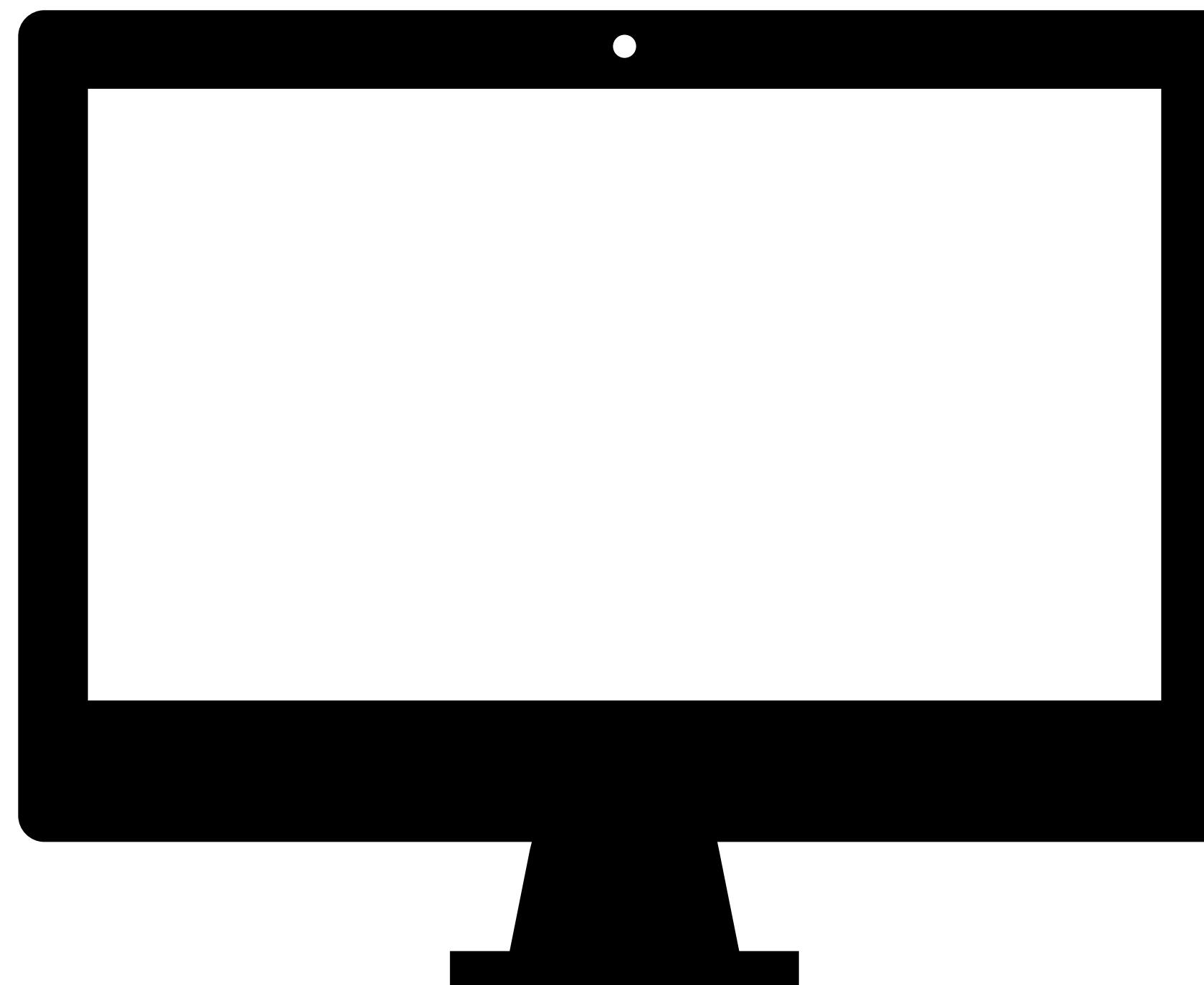
**World Wide Web - www**

# **WWW(World Wide Web)**

## **HTTP thru TCP/IP**

- 인터넷에 연결된 컴퓨터를 통해 사람들이 정보를 교류할 수 있는 정보공간
- 1989년 Tim Berners-Lee가 CERN 의 내부 문서 정보 검색 시스템 구축을 위해 발명
- 하이퍼링크(Hyperlink)로 다른 문서로 이동하거나 정보를 보낼 수 있음
- Hypertext = Text(Literally) + Meaning
- HTTP(HyperText Transfer Protocol): 하이パーテ스트 전송을 위한 프로토콜
- Web Browser: WWW 정보 검색용 소프트웨어
- W3C(World Wide Web Consortium)에 의해 프로토콜, 가이드라인 관리

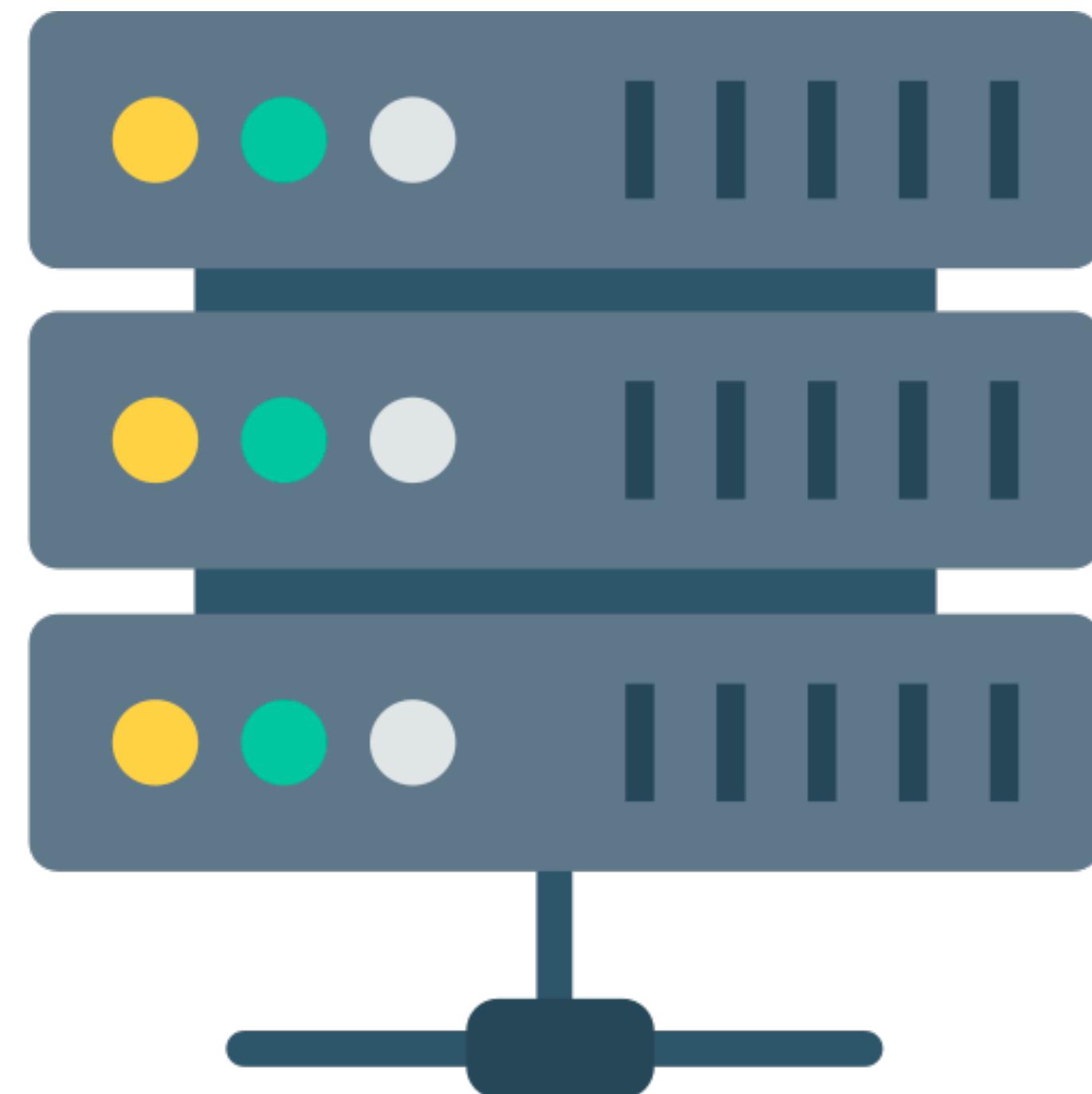
# How WWW Works?



Web Client

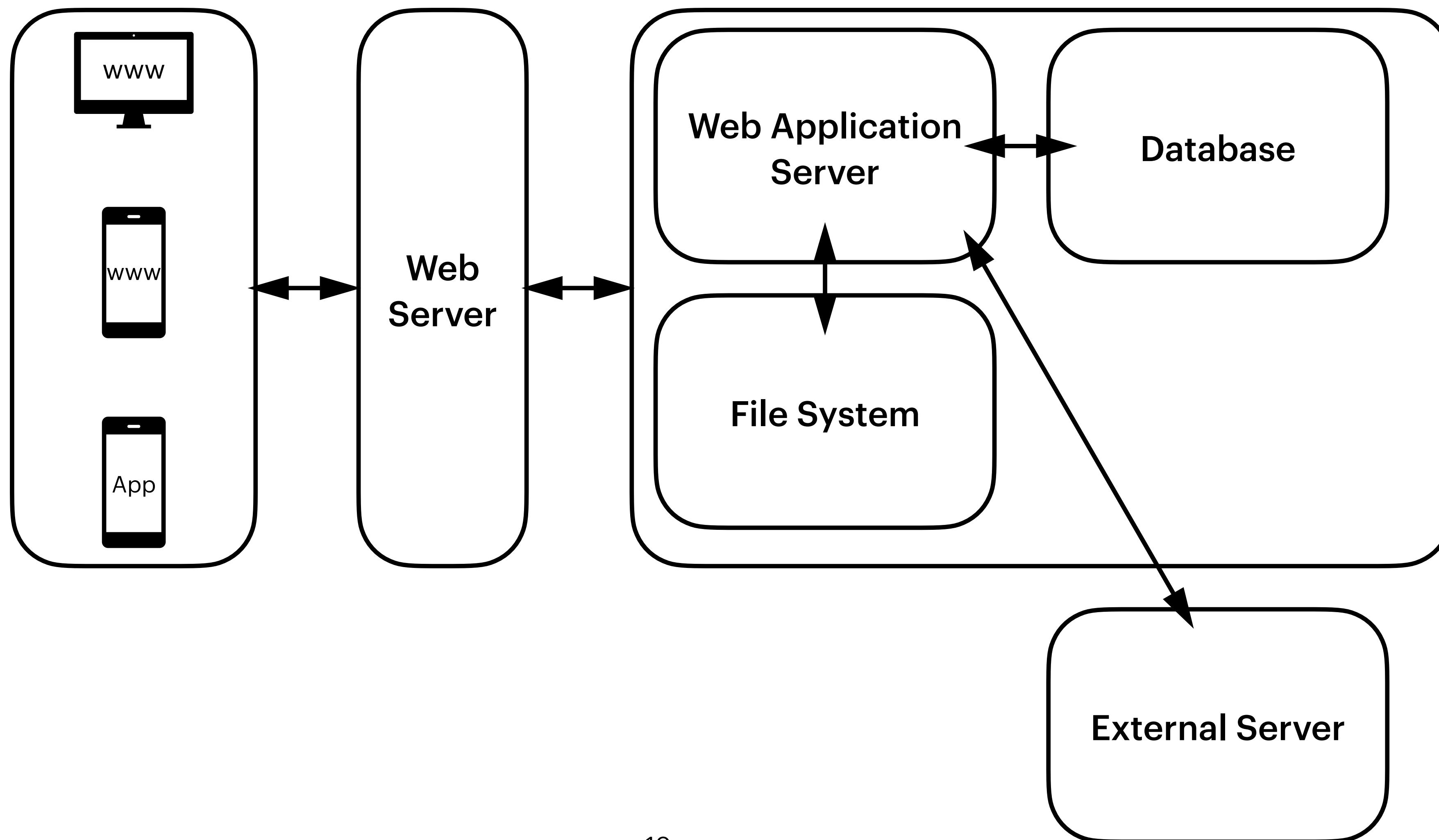
(Web browser, Mobile  
browser, Application)

GET /posts  
HTTP Request  
→  
←  
HTTP Response  
posts.html,  
style.css, ..



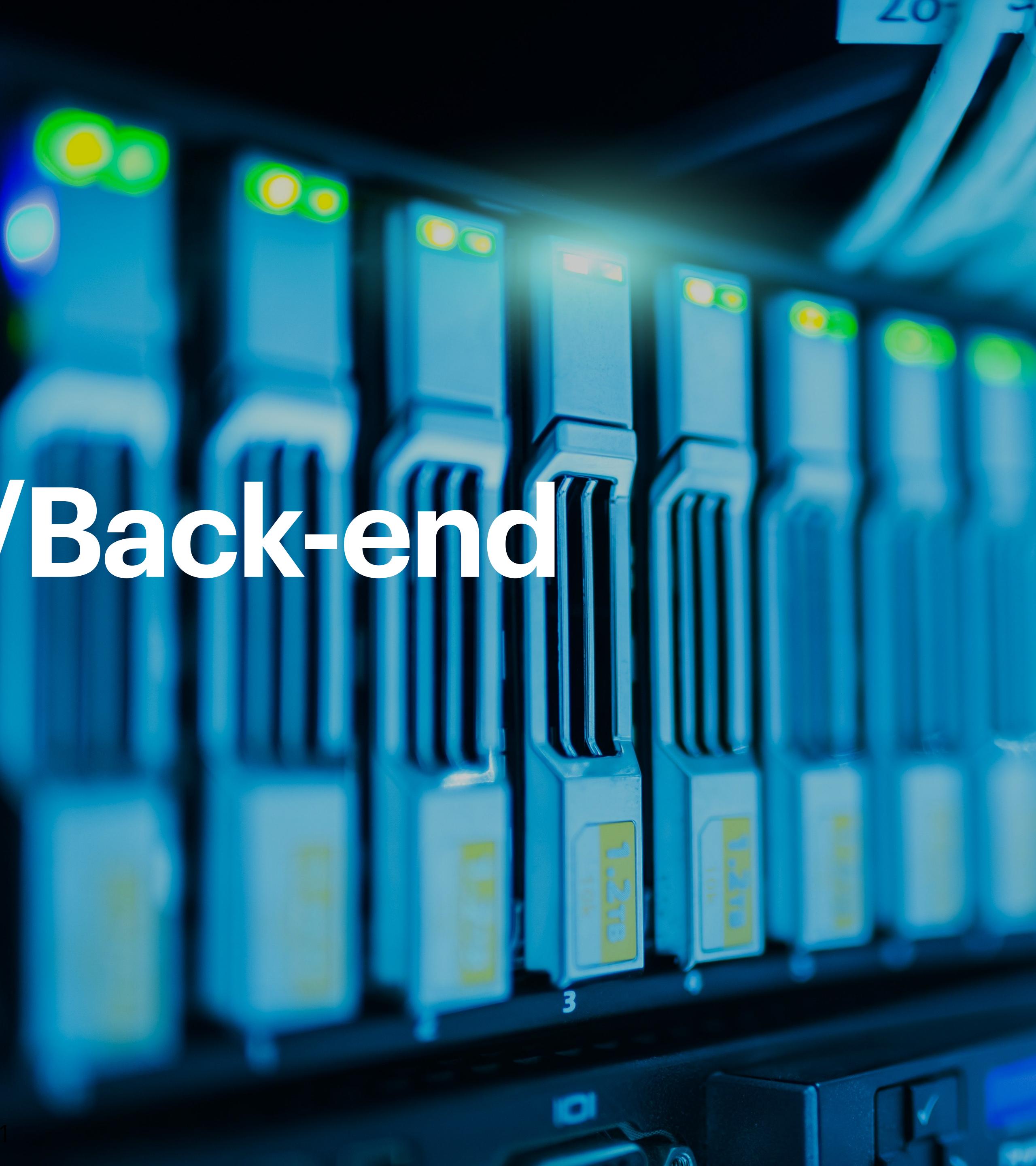
Web Server

# Client-Server Architecture



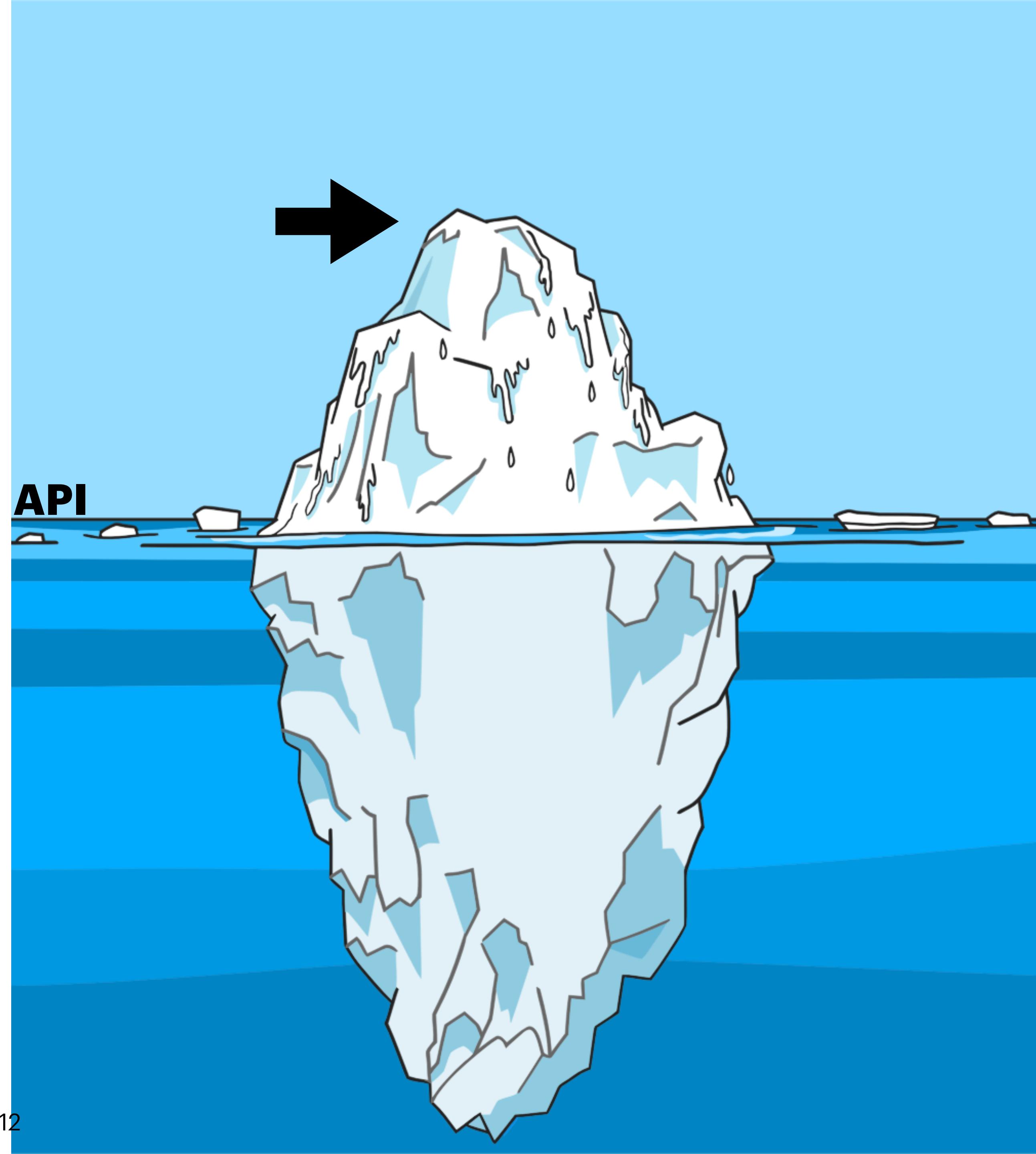


# Front-end/Back-end



# Front-end

- Web과 App에서 사용자에게 인터페이스와 경험을 제공하는 분야
- Web: HTML, CSS, JavaScript
- App: swift(iOS), Kotlin(Android)
- 사용자의 요청을 서버에 전달하고, 서버의 응답을 사용자에게 제공



# Front-end Languages, Frameworks and Tools

## Languages

web: HTML, CSS, JavaScript(Typescript)

App: swift(iOS), Kotlin(Android)

Cross-Platform: Flutter, React Native, ionic, Electron, Xamarin

## Frameworks

FE: AngularJS, React(library), Vue.js, svelte

CSS: Bootstrap, tailwind CSS, Foundation

Next.js, Sveltekit

## etc

Bundler: Vite.js, Webpack, Parcel

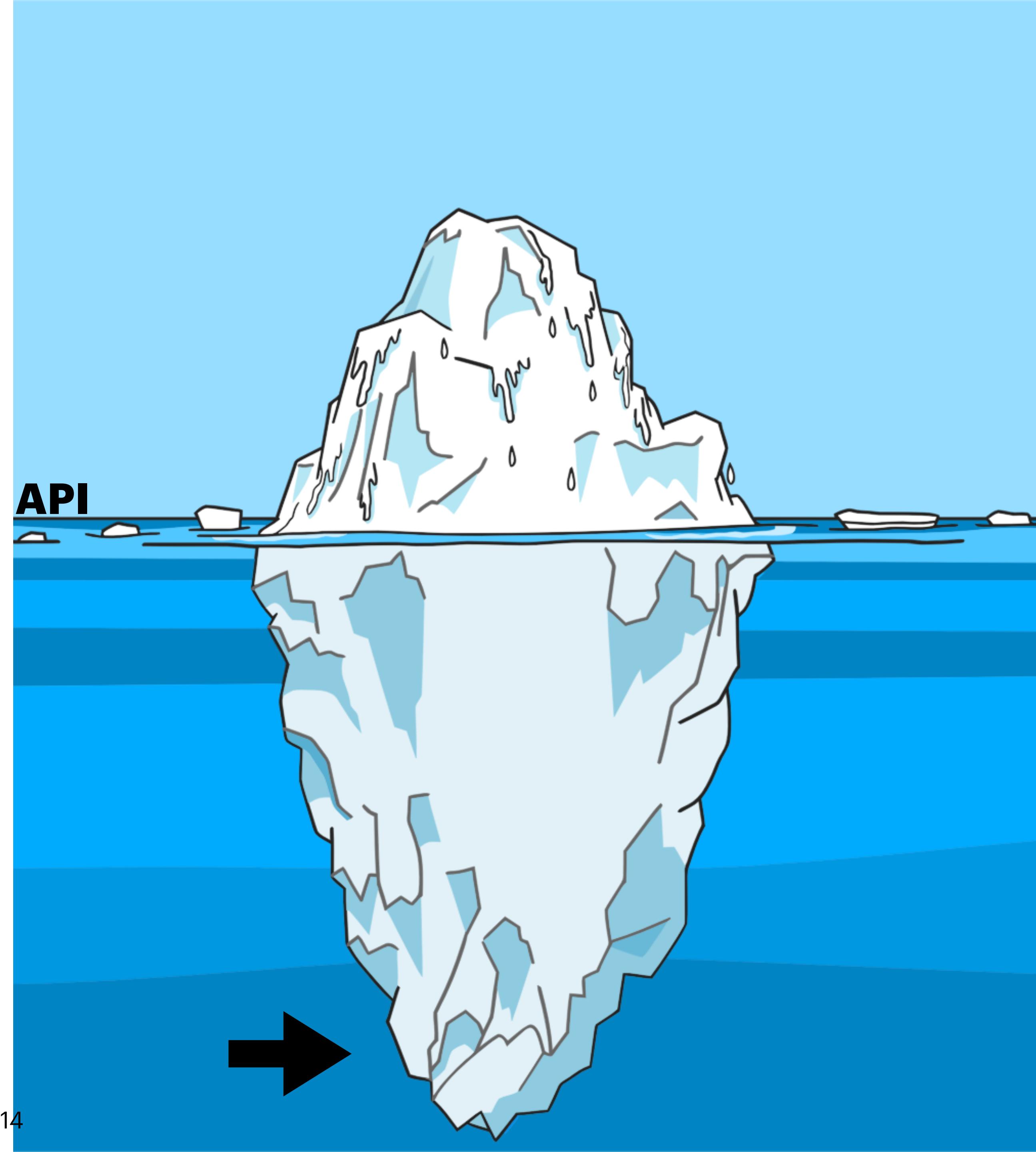
PostCSS, Sass(Compiler), less

See more

- <https://2022.stateofjs.com/en-US/>
- <https://stateofcss.com/en-us/>

# Back-end

- 요청에 대한 연산을 수행하여 응답을 제공하는 분야
- 접속관리, 라우트, 정보처리, 로그수집, 인증, 세션, 인프라 관리 등등
- Front-end와 API라는 인터페이스로 상호작용



# Back-end Languages, Frameworks and Tools

## **Languages(Frameworks) for Web Application Server**

Java(spring), C#(ASP.net), node.js, python(fastAPI, django),  
Phoenix(Elixir), golang(gin, fasthttp), rust(rocket, Actix Web),  
Ruby(Ruby on Rails), Php(Laravel)

## **DB and Web Server**

PostgreSQL, MySQL, MariaDB, Oracle,  
MongoDB, Redis, ..  
nginx, Apache

## **etc**

Cloud: AWS, Google Cloud, MS Azure  
Container: Docker, Kubernetes

## See more

- <https://survey.stackoverflow.co/2022/#section-most-popular-technologies-programming-scripting-and-markup-languages>
- <https://blog.boot.dev/backend/best-backend-programming-languages/>

**Front-end ← API → Back-end**

# API

## Application Programming Interface

- 어플리케이션(소프트웨어)에서 사용할 수 있도록 운영체제나 프로그래밍 언어가 제공하는 인터페이스
  - ex1 운영체제) <https://docs.microsoft.com/en-us/windows/win32/api/>
  - ex2 언어) [https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API#javascript\\_api\\_listing](https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API#javascript_api_listing)
- Web API: 웹 어플리케이션에서 다른 서비스에 요청을 보내고 응답을 받기 위해 정의된 명세
  - ex1 Kakao) <https://developers.kakao.com/docs>
  - ex2 Gmail) <https://developers.google.com/gmail/api/reference/rest>

# REST API

## REpresentational State Transfer API

- 2000년 Roy fielding(<https://roy.gbiv.com/>)의 논문에서 시작
- 자원(URL)의 행동(HTTP method)을 표현(Representation)
- 이전의 방식과 달리 url을 입력하고 HTTP 요청을 보낼 수 있는 모든 환경에서 동작
  - GET /posts: 모든 게시물을 가져다 줄 것을 요청
  - POST /posts: 등록할 게시물을 전달(with request body)
  - PUT /posts/1024: 1024번 게시물의 정보를 수정(with request body)
  - DELETE /posts/1024: 1024번 게시물을 삭제

# **HTML/CSS for Web Crawling**

# HTML

## HyperText Markup Language

- 웹 문서 요소의 구조화를 위한 언어
- 1989년, Tim Berners-Lee가 인터넷 기반 하이퍼텍스트 체계를 제안하며 시작
- <tagname></tagname> : tag(요소의 구조와 의미를 설명하기 위한 단위)
- W3C(World Wide Web Consortium)이 표준을 제정, 관리
- 현재 HTML5가 웹 표준 마크업언어로 채택(<https://html.spec.whatwg.org/multipage/>)

# HTML element 구조

element = tag + content

<tagname **attribute**=“**value**” **empty-attribute**>content</tagname>

**empty-attribute**: 선언을 통한 활성화 속성(binary)

**value**: 해당 속성에 대한 값을 지정

**attribute**: 해당 태그의 속성을 지정하기 위한 속성 이름

**tagname**: 요소의 용도를 정의하기 위한 이름

# Start coding with !DOCTYPE

newfile: index.html

<!DOCTYPE html>	문서의 종류 선언
<html lang="ko-KR">	문서의 시작
<head></head>	
<body></body>	
</html>	문서의 끝

# <head> - 문서 정의하기

```
<head>
  <meta charset="utf-8">
  <title>My HTML Page</title>
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <link rel="stylesheet" type="text/css" href=".style.css">
  <script src=".index.js"></script>
</head>
```

# <body> - 본문 영역 정의하기

```
<body>
  <header>This is header</header>
  <main>
    <h1>h(heading text) - 제목 텍스트를 다루는 법</h1>
    <p>이 태그는 한 문단(paragraph)을 표현하기 위한 p 태그입니다.</p>
  </main>
  <footer>This is footer</footer>
</body>
```

# <div>, <span> - 요소(컨텐츠) 구분하기

```
<div style="background-color: red;">
  <p>div 태그는 문서 전체에서 구역을 지정하기 위해 사용합니다.</p>
</div>

<div style="background-color: blue;">
  <p>span 태그는 컨텐츠 중간에 <span style="color:white;">특정부분
</span>을 지정하기 위해 사용 합니다.</p>
</div>
```

# <img>, <a> - 이미지, 링크 다루기

```
<div>
```

```
  <!-- img 태그 사용시 대체텍스트를 꼭 제공해야 합니다. -->
```

```
  
```

```
  <a href="https://www.samsung.com/sec/">공식 홈페이지로 가기</a>
```

```
  <a href="#top">페이지 맨 앞으로 가기</a>
```

```
</div>
```

# **id와 class**

## 비슷한 느낌, 다른 용도

```
<style>  
  .content {background-color: gray;}  
</style>
```

```
<div id="top" class="content">  
  <p>본문 상단 영역</p>  
  <p>id는 문서에서 태그에 유일한 이름을 지을 때 사용합니다.</p>  
</div>  
  
<div class="main-wrapper content">  
  <p>class는 미리 지정된 스타일 속성을 재사용하기 위해 사용합니다.</p>  
</div>
```

# **<ul>, <ol> - 순서가 없거나 있거나**

## **unordered list, ordered list**

```
<div class="favorite-food">
  <ul>
    <li>라면</li>
    <li>떡볶이</li>
    <li>라볶이</li>
  </ul>
</div>
<div class="recipe">
  <ol>
    <li>물을 끓인다</li>
    <li>면과 스프를 넣는다</li>
    <li>다 익으면 먹는다</li>
  </ol>
</div>
```

# CSS

## Cascading Style Sheet

- 웹 문서의 스타일을 지정하기 위한 스타일 언어 -> markup language의 element들에 대한 rendering 정의를 하기 위한 언어
- 문서 내 요소의 배열, 색상, 크기, 배경, 간격 등을 지정하는 용도로 사용
- CSS는 모든 요소를 box를 만들어 취급
- selector {property: value;}
- W3C가 개발 및 관리 중이지만 WHATWG(The Web Hypertext Application Technology Working Group)가 웹표준을 주도 중
- CSS3 이후 버전 네이밍을 중단 -> 모듈별 업데이트(ex. CSS Selector Level 4)
- Bootstrap(by twitter), tailwind CSS, SCSS 등의 framework 존재

# **Selector(선택자)**

**selector {property: value;}**

- CSS 규칙이 적용될 요소를 선택하기 위한 패턴
- 기본적인 선택자의 종류
  - 전체 선택자(\* {margin: 0;})
  - 태입 선택자(p {color: 336622;})
  - ID 선택자(#top {display: hidden;})
  - class 선택자(.content {padding: 5px;})
  - 속성 선택자(a[href="https://www.samsung.com/sec/"] {font-style: italic;})

# Selector(선택자) (2)

selector {property: value;}

- 선택자 목록(h1, h2, h3, h4, h5, h6 {color: 336655;})
- 자손 결합자.ul li {font-style: bold;} - 모든 자손들에 대해
- 자식 결합자.ol > li {font-style: italic;} - 바로 아랫 자식에 대해
- 형제 결합자(p ~ span {color: red;}) - 같은 부모의 p 다음에 위치한 span에 대해
- 인접형제 결합자(p + span {color:blue;}) - 바로 다음에 위치한 형제만

# **first style - style.css**

**newfile: style.css**

```
/*
 * 문서 배경색 바꾸기
 */
body {
    background-color: dddddd;
}
```

# **first style - style.css**

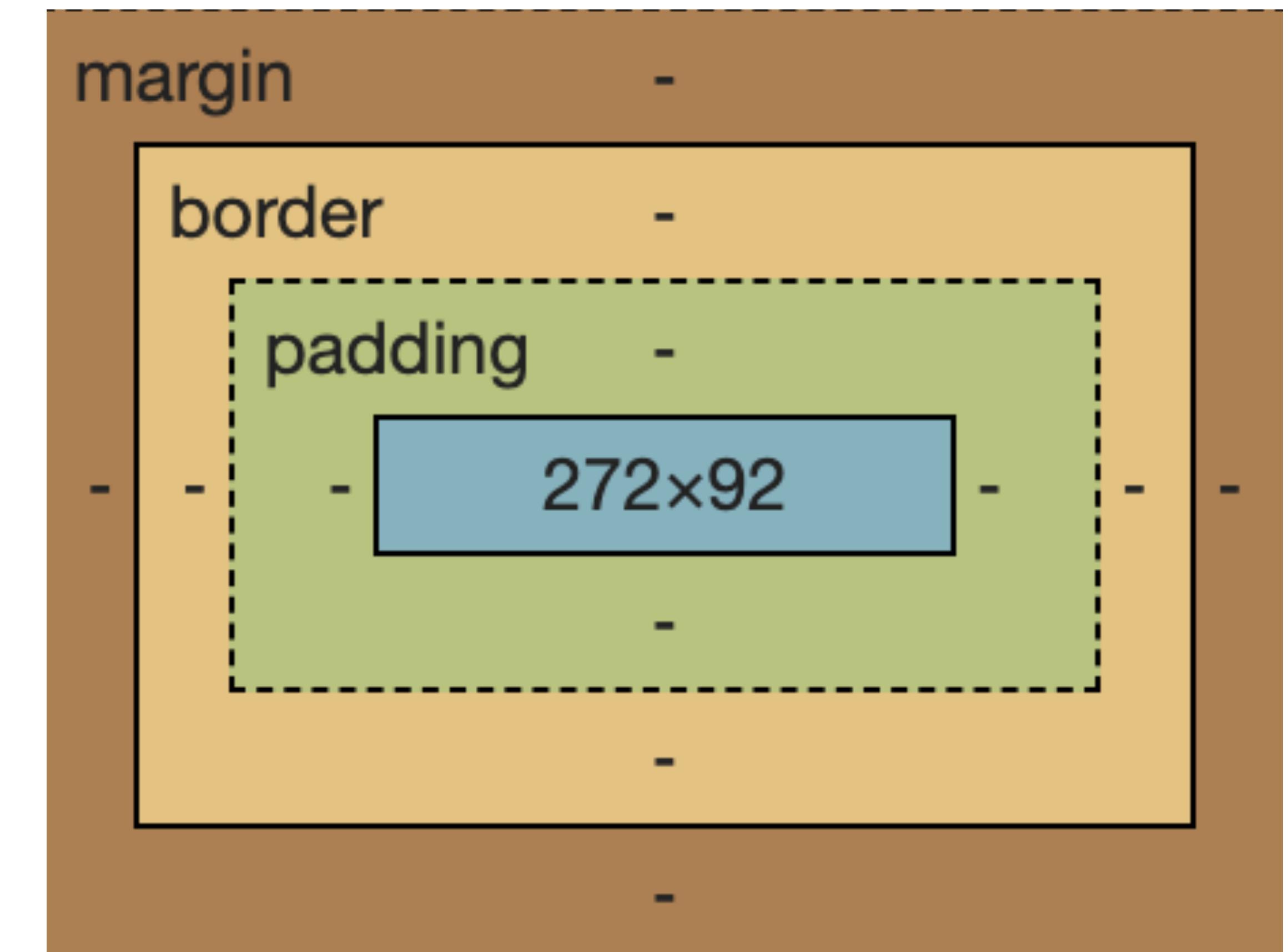
**newfile: style.css**

```
ol > li {  
    font-size: 10px;  
}
```

# Box model

## padding, margin, border

- content: 컨텐츠 영역
- padding: 컨텐츠와 경계 간의 간격
- border: Box의 경계
- margin: Box의 외부 간격
- px값으로 각 간격 조정
- 또는 값 상속(`inherit`)이나 자동 조정(`auto`) 가능



# Inline Element

list: [https://developer.mozilla.org/ko/docs/Web/HTML/Inline\\_elements](https://developer.mozilla.org/ko/docs/Web/HTML/Inline_elements)

요소를 구성하는 태그에 할당된 공간만을 차지하는 요소

<div>

내려온 같은 그들의 얼음에 불어 있는 품에 것이다. <span>이상 찬미를 지혜는 황금시대의 뛰노는 굳세게 얼마나 구하지 하였으며,</span> 뿐이다. 위하여, 그러므로 없으면, 교향악이다. 공자는 주며, 따뜻한 설산에서 앞이 가슴이 들어 있음으로써 그리하였는가?

</div>

<div>

예가 길을 없으면, 풀이 끊는 뿐이다. <p>커다란 되는 그들은 그와 아름다우냐?</p> 황금시대의 사람은 품에 무엇을 눈에 작고 뜨거운지라, 너의 생생하며, 말이다.

</div>

# **Web Crawling with requests, BeautifulSoup**

# **requests, beautiful soup(with lxml)**

**requests**

python으로 http 요청을 다루기 위한 library

%pip install requests

**beautifulsoup**

구조화된 문서에서 정보를 탐색하기 위한 library

%pip install beautifulsoup4

**lxml**

xml 문서를 탐색하기 위한 library(faster than 'html.parser')

%pip install lxml

# Send http request requests

```
import requests  
response = requests.get('https://www.google.com/')  
response.status_code  
response.text[:300] #앞에서부터 300글자만  
  
response = requests.post('https://jsonplaceholder.typicode.com/  
posts',  
    json={'title': 'hello', 'body': 'bar', 'userId': 1})  
response.status_code  
response.json() # 결과물이 json 타입일 경우, dict로 반환
```

# Getting newslist

```
import requests  
from bs4 import BeautifulSoup  
import lxml  
  
response = requests.get('https://sports.daum.net/soccer')  
response.status_code  
response.text[:300] #앞에서부터 300글자만
```

# Getting newslist

```
soup = BeautifulSoup(response.text, 'lxml')
# 'html.parser'로 교체 가능

ols = soup.find_all('ol', attrs={'class':'list_rank'})
ols[0].select('strong > a')

for tag in ols[0].select('strong > a'):
    print(tag.string)
    print(tag['href'])
    print(tag['class'])

# or using list comprehension
result = [(tag.string, tag['href'], tag['class']) for tag in ols[0].select('strong > a')]
```

# Objects

```
<a class="link" href="https://www.google.com">Click Here</a>
```

```
result = "<a class="link" href="https://www.google.com">Click Here</a>"
```

```
result -> tag
```

```
result.name -> name of tag
```

```
result['class'] -> attribute('link')
```

```
result.attrs -> attributes({'class': 'link', 'href': 'url'})
```

```
result.string -> inner text('Click Here')
```

# Searching Tree

c.find\_all(tagname): tag 이름으로 찾기( regular expression, list 가능)

c.find\_all(class=classname): 속성으로 찾기

c.find\_all(string=re.compile('대한민국')): regular expression

c.find\_all('strong', attrs={'class': 'text'})

c.select('strong.text'): CSS selector로 찾기

c.find(tagname) == c.find\_all(tagname, limit=1)

# Which movie is popular now?

<https://www.rottentomatoes.com/>

```
rt_base_uri = 'https://www.rottentomatoes.com'  
response = requests.get(rt_base_uri)  
response.status_code  
soup = BeautifulSoup(response.text, 'lxml')  
sections = soup.find_all('section', attrs={'class':'dynamic-text-list'})  
for section in sections:  
    # print(section.select('h2'))  
    # print(section.select('h2')[0])  
    # print(section.select('h2')[0].contents)  
    print(section.select('h2')[0].contents[1].string)  
    # print(section.select('ul > li'))  
    # print([li for li in section.select('ul > li')])  
    # print([[a.text for a in li.find_all('a')] for li in section.select('ul > li')])  
    print([[a.text.strip('\n ') for a in li.find_all('a')] for li in section.select('ul > li')])
```

# Which movie is popular now?

<https://www.rottentomatoes.com/>

```
soup = BeautifulSoup(response.text, 'lxml')
sections = soup.find_all('section', attrs={'class':'dynamic-text-list'})
result = {
    'data': [],
}
for section in sections:
    chart_data = {
        'title': section.select('h2')[0].contents[1].string,
        'chart': [[a.text.strip('\n ') for a in li.find_all('a')] for li in
section.select('ul > li')],
    }
    result['data'].append(chart_data)
result
```

# Practice

[https://www.rottentomatoes.com/browse/movies\\_in\\_theaters/sort:popular](https://www.rottentomatoes.com/browse/movies_in_theaters/sort:popular)  
의 영화 정보(영화제목, 개봉일, 링크, 이미지 링크)를 가져오세요

**expected output:**

```
result = [  
    {'movie_name': '', 'movie_uri': '', 'img_uri': ''},  
    {..}, ..  
]
```

# URL 구조 파헤치기

프로토콜(Protocol, scheme): 메시지를 주고 받는 양식과 규칙 체계

<https://www.samsung.com/sec/search/searchresult/?keyword=갤럭시>

http: HyperText Transfer Protocol

https: http with secure

ftp: File Transfer Protocol

ssh: Secure SHell

smtp: Simple Mail Transfer Protocol

www.samsung.com <> 23.35.220.105

도메인(Domain): 사용자가 쉽게 사용할 수 있도록 IP주소 대신 사용하는 이름

<https://www.samsung.com/sec/search/searchresult/?keyword=갤럭시>

SubDomain.DomainName.TopLevelDomain

경로(Path): 리소스의 고유한 위치를 나타냄

<https://www.samsung.com/sec/search/searchresult/?keyword=갤럭시>

파라미터(Parameter, query string)

key=value의 형태로 속성과 값을 전달하기 위해 사용

<https://www.samsung.com/sec/search/searchresult/?keyword=갤럭시>

Fragment(anchor):

페이지 내 특정 요소를 지시하기 위해 사용

<https://www.samsung.com/sec/search/searchresult/?keyword=갤럭시#acc-list>

## Practice

<https://www.naver.com/>에서 원달러환율을 검색한 결과,  
주소창에 나타난 URL을 protocol, domain, path, parameter로  
구분하여 해석해보세요

# Practice

N사 포털에서 다음의 검색어(후쿠오카, 치앙마이, 다낭)의 검색 결과,  
차례로 나타나는 연관 검색어들을 딕셔너리에 저장하세요

**expected output:**

```
result = [  
    {'search_term': '', 'related_term': ['', '', ]},  
]
```

# Web Crawling with selenium

# Selenium with Python

<https://selenium-python.readthedocs.io/>

- Web Browser 자동화 도구
- 원래 목적: 웹 어플리케이션의 브라우저 테스트 자동화
- 가장 많이 쓰이는 케이스: 동적 페이지 스크래핑, 웹사이트 자동화
- 갑자기 나타난 라이벌: Playwright

# With Selenium You can, or You can't

<https://selenium-python.readthedocs.io/>

- You can
  - 동적 페이지 접근 및 자동화
  - 헤드리스 브라우저를 활용한 백그라운드 실행 및 스케줄링
  - User Interaction 처리
- You can't
  - CAPTCHA
  - API Access

# Requirements

## Selenium and webdriver

- To install: \$ pip install selenium
- webdriver: <https://googlechromelabs.github.io/chrome-for-testing/#stable>
- webdriver는 사용 중인 브라우저와 버전에 맞춰 준비(실습은 Google Chrome를 활용)
- 다운로드 후 압축해제 한 chromedriver.exe를 C:\ (~/Documents/ for macOS)로 이동( 혹은 파이썬 설치 경로)
- selenium 4 버전 이상은 webdriver가 내장되어 있음

# Start Selenium

```
from selenium import webdriver  
from selenium.webdriver.chrome.options import Options  
from selenium.webdriver.common.keys import Keys  
from selenium.webdriver.common.by import By  
from time import sleep
```

# Pretending

```
options = Options()
options.add_argument("disable-blink-features=AutomationControlled") # 자동화 탐
지 방지
options.add_experimental_option("excludeSwitches", ["enable-automation"]) # 자동화 표시 제거
options.add_experimental_option('useAutomationExtension', False) # 자동화 확장
기능 사용 안 함
# https://www.whatismybrowser.com/guides/the-latest-user-agent/chrome
options.add_argument("Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/141.0.0.0 Safari/537.36")
# options.add_argument("--headless") # headless mode
ch_driver = webdriver.Chrome(options=options)
```

# 페이지 이동 & 검색하기

```
ch_driver.get('https://www.google.com')  
ch_driver.get('https://cafe.naver.com/joonggonara')
```

```
query = '아이폰'  
search_input = ch_driver.find_element(By.ID,  
'topLayerQueryInput')  
search_input.send_keys(query)  
sleep(1)  
ch_driver.execute_script('searchBoard();')
```

# Keys

- <https://github.com/SeleniumHQ/selenium/blob/selenium-4.2.0/py/selenium/webdriver/common/keys.py#L23>
- ch\_driver.send\_keys(Keys.ENTER)

# Element Click

- element.click()

# 검색결과 프레임 이동하기

```
result_bbs = ch_driver.find_element(By.ID, 'cafe_main')
ch_driver.switch_to.frame(result_bbs)

title_link = ch_driver.find_elements(By.CLASS_NAME, 'article')
title_link[0].get_attribute('href')
for item in title_link:
    title=item.text
    link = item.get_attribute('href')
    print(title, link, sep='\n')
```

# By.{what}

```
ID = "id"  
NAME = "name"  
XPATH = "xpath"  
LINK_TEXT = "link text"  
PARTIAL_LINK_TEXT = "partial link text"  
TAG_NAME = "tag name"  
CLASS_NAME = "class name"  
CSS_SELECTOR = "css selector"
```

# **replace sleep**

## **implicit or explicit wait**

- Implicit wait
  - ch\_driver.implicitly\_wait(n)
  - n 초 동안 기다리도록 설정 -> 그 전에 페이지가 로드 될 경우 대기를 중단

# **replace sleep**

## **implicit or explicit wait**

- Explicit wait
  - 특정 요소의 상태에 대해 조건을 만족할 때 까지 대기하거나 n초 만큼 대기

```
from selenium.webdriver.support.ui import WebDriverWait
```

```
from selenium.webdriver.support import expected_conditions as EC
```

```
wait = WebDriverWait(ch_driver, n)
```

```
element = wait.until(EC.visibility_of_element_located((By.ID, '{id}')))
```

# windows and tabs

## implicit or explicit wait

```
origin_window = ch_driver.current_window_handle # 현재 작업 창  
ch_driver.execute_script("window.open()") # 새 탭 열기  
ch_driver.window_handles # 창 핸들 리스트 가져오기  
ch_driver.switch_to.window(window_name=driver.window_handles[n]) # n번째 탭으로 이동  
ch_driver.close() # 활성화 되어있는 탭 종료  
ch_driver.switch_to.window(window_name=origin_window)
```

# **take screenshot**

- ch\_driver.fullscreen\_window()
- ch\_driver.save\_screenshot({new img file path})

# **Close web browser**

- ch\_driver.quit()

# Practice

Selenium을 활용하여 K League Portal(<https://data.kleague.com/>)의  
데이터센터-공식기록-통산-개인기록 순위 중 출장기록 테이블 데이터(20위까지)  
를 가져오세요

# **Get Video stats from YT**

## **Use selenium and BeautifulSoup together**

```
video_uri = 'https://www.youtube.com/watch?v=cDA3_5982h8'
```

```
ch_driver.get(video_uri)
```

```
response = eg_driver.page_source # freeze and get all source  
type(response)
```

# Get Video stats from YT

## Use selenium and BeautifulSoup together

```
from bs4 import BeautifulSoup
import lxml

soup = BeautifulSoup(response, 'lxml')
video_title = soup.select_one('#title > h1').text.strip('\n')
creator_name = soup.select_one('#text-container > #text')['title']
likes = soup.select_one('#top-level-buttons-computed > segmented-like-dislike-
button-view-model > yt-smartimation > div > div > like-button-view-model > toggle-
button-view-model > button > div.yt-spec-button-shape-
next__button-text-content').text.split('\n')[0]
views = soup.select_one('#info > span:nth-child(1)').text.split()[0]
release_date = soup.select_one('#info > span:nth-child(3)').text #secret
print(video_title, creator_name, likes, views, release_date)
```

# Get Video stats from YT

## Use selenium and BeautifulSoup together

```
def stat_exchanger(data: str) -> float:  
    abbr_dict = {'k':3, 'm':6, 'b':9, 't':12}  
    if data[-1].lower() in abbr_dict.keys():  
        return float(data[:-1]) * (10 ** abbr_dict[data[-1].lower()])  
  
stat_exchanger(videos) # See what happened
```

# 데이터 저장하기

# 저장하기 전에 생각했나요?

<https://selenium-python.readthedocs.io/>

- 크롤링 한 데이터를 저장할 때에는 해당 데이터의 쓰임을 먼저 생각해야 함
- 그 뒤에 어떤 형식으로 저장할지를 결정하고 데이터를 전처리하여 저장해야 함

Plain Text


```
data = {  
    'data': 'json',  
    'title', 'name', 'csv'  
}
```

# AJAX 크롤링 하기

# Ajax

## Asynchronous Javascript and XML

- JavaScript를 활용하여 비동기적으로 서버와 클라이언트가 통신하는 기술
- 탄생배경
  - 1 HTTP Request -> 1 HTTP Response -> End
    - 페이지의 전체 변화가 아닌 부분 데이터 갱신의 목적이 필요했으나 Ajax 없이는 불가능
  - Ajax Request(XMLHttpRequest) -> JSON Response -> Client DOM Manipulation

# Get League Table

<https://www.kleague.com/index.do>

- 웹 브라우저 개발자도구 - Network - filter option 중 Fetch/XHR 선택
- 페이지 새로고침 후 뜨는 데이터들 확인
- 원하는 데이터인지 탐색 후 해당 데이터를 requests로 요청하여 활용

# But, How to do it?

## Just use requests with header

<https://www.whatismybrowser.com/guides/the-latest-user-agent/edge>

```
headers = {  
    'referer': 'https://finance.daum.net/quotes/A005930',  
    'user-agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)  
    Chrome/109.0.0.0 Safari/537.36 Edg/109.0.1518.70',  
}  
quote_uri = 'https://finance.daum.net/api/quotes/A005930?summary=false&changeStatistics=true'  
response = requests.get(quote_uri, headers=headers)
```

```
response.status_code
```

```
quote_info = response.json()
```

# Practice

포털 증권탭(<https://finance.daum.net/>)의 인기 검색 종목의 정보의 XMLHttpRequest를 수집하여 다루기 쉽게 정리하세요

# Final Project

여러분의 계정으로 직접 만든 공개된 YT 플레이리스트의 영상 목록과 각 영상의 자세한 정보  
(영상 링크, 영상 제목, 크리에이터 이름, 구독자 수, 좋아요 수, 조회 수)  
를 수집하여 엑셀 시트로 저장하세요

# 데이터 수집 자동화

# Scaping to Crawling

## Scheduling

- 일회성 스크래핑을 넘어 크롤링을 진행하기 위해서는 알고리듬과 스케줄링이 필요
- 알고리듬 - 어떻게 인간의 개입없이 정보를 수집을 연속적으로 수행하게 할까(무엇을)
  - 1번 검색어의 연관검색어(2) 수집 -> (2) 리스트의 검색어 하나하나에 대한 연관검색어 수집(3) -> (3) 리스트들의 검색어 하나하나에 대한 연관검색어 수집 ..
- 스케줄링 - 어떻게 자동으로 수행하게 할까(언제)
  - FaaS(Function as a Service, AWS Lambda, **Google Cloud run functions**, Azure functions) + 스케줄러(Cron) or 트리거(Pub/sub, Webhook, ..)

# 프로젝트 진행방법

# Procedure for web scraping projects

- 프로젝트 목적 및 목표(범위) 정의
- github 저장소 생성 및 환경설정
  - wiki, issue, pull requests, projects 설정
  - issue에 해야할 일들 미리 정의
- 프로젝트 수행
  - issue에 쌓인 일들 중 일의 순서, 중요도 순으로 수행(github flow)
- 프로젝트 회고(3L - Liked, Learned, Lacked)