AI Programming

Python

04. Python Basic 4

first coding

데이터 수집 개요 및 HTTP 요청

데이터 수집 방법 개요

HTTP 요청과 응답 이해 (GET, POST 등)

requests 모듈을 활용한 데이터 수집

데이터 수집 방법 개요

- **데이터 수집 방법 개요**

  - 데이터 수집은 데이터를 효율적으로 모으고 활용하기 위해 중요한 단계
  - 데이터는 다양한 형식과 출처에서 수집될 수 있으며, 목적에 따라 방법도 달라짐
    → 목적과 환경에 따라 적합한 방법을 선택해야 함

데이터 수집 방법 개요

- **데이터 수집 방법**

① **웹 스크래핑(Web Scraping)**

- 웹 스크래핑은 웹사이트에서 HTML 데이터를 가져와 필요한 정보를 추출하는 방법
- 자동화해서 데이터를 수집할 수 있음
- 예시
  - 목적: 온라인 쇼핑몰의 상품 가격 정보 수집
  - 방법
    1. BeautifulSoup과 같은 Python 라이브러리를 사용
    2. 웹 페이지의 HTML을 파싱
    3. 상품명과 가격을 가져옴.

데이터 수집 방법 개요

- 데이터 수집 방법

② API 활용

- API(Application Programming Interface)는 데이터를 쉽게 요청하고 응답 받을 수 있도록 제공되는 인터페이스 → 주로 REST API를 통해 JSON 형태의 데이터를 제공
- 예시
  - 목적: 날씨 데이터를 수집
  - 방법
    - » requests 모듈을 이용하여 OpenWeatherMap API를 호출
    - » 공공데이터 포털 (https://www.data.go.kr/)
    - » 지방행정인허가데이터개방: LOCALDATA (https://www.localdata.go.kr/)

데이터 수집 방법 개요

- **데이터 수집 방법**

③ 데이터베이스에서 데이터 가져오기

- 이미 수집된 데이터를 데이터베이스에서 쿼리를 통해 가져오는 방법
  → RDBMS(SQL)이나 NoSQL 데이터베이스를 사용
- 예시
  - 목적: 고객 정보 조회
  - 방법
    - » RDBMS를 이용하여 select를 이용해서 데이터를 조회
      → 데이터베이스 관련 모듈 이용

데이터 수집 방법 개요

- **데이터 수집 방법**

④ 공개된 데이터셋 활용

- 정부나 기업, 연구기관에서 제공하는 공개 데이터셋을 다운로드하거나 API를 통해 사용
  → Kaggle, Google Dataset Search 등이 자주 사용
- 예시
  - 목적: 인구 통계 데이터 분석
  - 방법
    - » Kaggle에서 데이터를 다운로드한 뒤 Pandas로 읽어 분석

데이터 수집 방법 개요

- **데이터 수집 방법**

⑤ **IoT 또는 센서 데이터 수집**

- 센서를 이용해 실시간으로 데이터를 수집합니다
  → 온도 센서, 카메라, GPS 장치 등이 사용
- 예시
  - 목적: 스마트 홈의 온도 데이터를 수집
  - 방법
    - » MQTT 프로토콜을 이용하여 센서 데이터를 수집

데이터 수집 방법 개요

- **데이터 수집 방법**

⑥ 소셜 미디어 데이터 수집

- 소셜 미디어 플랫폼에서 사용자 활동 데이터를 수집하는 방법
  → 해당 플랫폼의 API를 사용
- 예시
  - 목적: 트위터의 특정 키워드로 트윗 수집
  - 방법
    - » Tweepy 라이브러리를 사용하여 트위터 API 호출

데이터 수집 방법 개요

- **데이터 수집 방법**
  - 웹 스크래핑: HTML 데이터 파싱
  - API 활용: JSON 데이터 가져오기
  - 데이터베이스: 저장된 데이터 검색
  - 공개 데이터셋: Kaggle 등에서 다운로드
  - IoT 센서: 실시간 데이터 수집
  - 소셜 미디어: 트위터 API 활용

requests 모듈을 활용한 데이터 수집

- **requests 모듈을 활용한 데이터 수집**

  - Python의 requests 모듈은 HTTP 요청을 보내고 응답을 처리하는 데 사용되는 라이브러리
  - 웹 서버와 통신하여 데이터를 가져오거나 전송하는 작업을 쉽게 수행
    - 다양한 HTTP 메서드(GET, POST, PUT, DELETE 등)를 지원
    - 데이터를 요청하고 응답을 처리하는 과정을 간단하게 처리

requests 모듈을 활용한 데이터 수집

- requests 모듈을 활용한 데이터 수집

  - requests 모듈 설치

```
pip install requests
```

requests 모듈을 활용한 데이터 수집

- **requests 모듈을 활용한 데이터 수집**

  - HTTP 요청의 주요 메서드
    - GET 요청

      서버에서 데이터를 가져오는 데 사용
    - **목적** : 웹 페이지의 내용을 가져오기

```
import requests

url = 'https://jsonplaceholder.typicode.com/posts'
response = requests.get(url)

if response.status_code == 200:
    # 상태 코드 200은 성공을 의미
    print("데이터 가져오기 성공!")
    print(response.text[:500])
    # 응답 본문의 일부 출력
else:
    print(f"오류 발생: {response.status_code}")
```

```
--
```

requests 모듈을 활용한 데이터 수집

- requests 모듈을 활용한 데이터 수집

  - 응답 데이터 처리
    - response.text
      - 응답 본문을 문자열로 반환
    - response.json()
      - JSON 형식의 응답을 딕셔너리로 변환
    - response.status_code
      - HTTP 상태 코드 반환

```
import requests

url =
'https://jsonplaceholder.typicode.com/posts/1'
response = requests.get(url)

if response.status_code == 200:
    data = response.json()
    # JSON 응답을 딕셔너리로 변환

    print(f"제목: {data['title']}")
    print(f"내용: {data['body']}")
else:
    print(f"오류 발생: {response.status_code}")
```

--

requests 모듈을 활용한 데이터 수집

- requests 모듈을 활용한 데이터 수집

  - 쿼리 파라미터 사용하기
    - 서버에 데이터를 전달할 때
      URL에 파라미터를 포함해서 전달
      → 특정 데이터 가져오기

```
import requests

url = 'https://jsonplaceholder.typicode.com/posts'
params = {'userId': 1}
# userId가 1인 게시글 가져오기

response = requests.get(url, params=params)

if response.status_code == 200:
    print("사용자 게시글 가져오기 성공!")
    print(response.json())
else:
    print(f"오류 발생: {response.status_code}")
```

```
--
```

requests 모듈을 활용한 데이터 수집

- **requests 모듈을 활용한 데이터 수집**

  - 실제 응용 사례
    - 웹 페이지 크롤링: 뉴스 헤드라인 가져오기
    - API 활용: 날씨, 주식 정보, 소셜 미디어 데이터 수집
    - 자동화: 특정 시간에 정기적으로 데이터 요청

  - 주의사항
    - API 호출 시 요청 제한(Request Limit)을 초과하지 않도록 주의
    - 민감한 데이터(API 키, 인증 정보)는 환경 변수로 관리하거나 안전하게 저장
    - 웹 스크래핑 시 사이트의 robots.txt 규칙을 준수

first coding

웹 스크래핑 기초

HTML 구조 이해 (태그, 속성, DOM)

BeautifulSoup을 활용한 웹 스크래핑

HTML 구조 이해 (태그, 속성, DOM)

- HTML이란?

  - HTML (HyperText Markup Language)은 웹 페이지의 구조를 정의하는 마크업 언어
  - HTML 문서는 브라우저가 읽어서 화면에 표시하는 기본 구조를 제공
  - HTML
    - **태그(tag)**를 사용하여 요소를 정의
    - 각 요소는 **속성(attribute)**과 콘텐츠를 포함
    - 웹 페이지의 텍스트, 이미지, 링크, 버튼 등 다양한 요소를 구성

HTML 구조 이해 (태그, 속성, DOM)

- 태그 (Tags)

  - 태그는 HTML 요소를 정의하며, 다음과 같은 형식

```
<태그이름 속성="값">내용(Contents)</태그이름>
```

  - **시작 태그**: <태그이름>
  - **종료 태그**: </태그이름> (일부 태그는 종료 태그가 없음. 예: <img>)
  - **내용**: 시작 태그와 종료 태그 사이에 들어가는 컨텐츠 데이터

```
<p>Hello, World!</p>
```

  - <p>: 문단(paragraph)을 나타내는 태그
  - Hello, World!: 태그 사이의 내용

HTML 구조 이해 (태그, 속성, DOM)

- 속성 (Attributes)

  - 속성은 태그에 추가 정보를 제공
  - 속성="값" 형식으로 작성, 속성은 시작 태그에만 추가
    - id: 요소의 고유 식별자
    - class: CSS 스타일링이나 JavaScript 제어를 위한 클래스 이름
    - href: 링크를 나타냄 (예: <a> 태그에서 사용)
    - src: 이미지나 외부 리소스의 경로 (예: <img> 태그에서 사용)

```
<a href="https://example.com" target="_blank">Click Here</a>
```

HTML 구조 이해 (태그, 속성, DOM)

- DOM (Document Object Model)

  - DOM은 브라우저가 HTML 문서를 파싱하여 생성한 객체 모델
  - HTML 문서를 트리 구조로 표현하며, 각 요소는 노드(Node)로 구성
  - DOM의 구성 요소
    - HTML 요소 (Element Nodes): 태그들
    - 속성 (Attribute Nodes): 태그의 속성들
    - 텍스트 (Text Nodes): 태그 안의 내용

```
<html>
  <body>
    <h1>Hello, World!</h1>
    <p>This is a paragraph.</p>
  </body>
</html>
```

```
html
└── body
    ├── h1
    │   └── "Hello, World!"
    └── p
        └── "This is a paragraph."
```

HTML 구조 이해 (태그, 속성, DOM)

- DOM (Document Object Model)

```
<!DOCTYPE html>
<html>
<head>
    <title>Sample Page</title>
</head>
<body>
    <h1 id="main-title">
        Welcome to Web Scraping
    </h1>
    <p class="description">
        This is an example paragraph.
    </p>
    <a href="https://example.com" target="_blank">
        Visit Example
    </a>
</body>
</html>
```

```
html
├── head
│   └── title
│       └── "Sample Page"
└── body
    ├── h1 (id="main-title")
    │   └── "Welcome to Web Scraping"
    ├── p (class="description")
    │   └── "This is an example paragraph."
    └── a (href="https://example.com", target="_blank")
        └── "Visit Example"
```

BeautifulSoup을 활용한 웹 스크래핑

- BeautifulSoup을 활용한 웹 스크래핑

  - Python에서 HTML과 XML 문서를 파싱하고 원하는 데이터를 추출할 때 사용하는 라이브러리
    → HTML의 DOM 구조를 탐색하고 태그, 속성, 텍스트를 쉽게 추출
  - 설치

    pip install beautifulsoup4

BeautifulSoup을 활용한 웹 스크래핑

- BeautifulSoup을 활용한 웹 스크래핑

```
from bs4 import BeautifulSoup

html_doc = """
<html>
  <head><title>Example Page</title></head>
  <body>
    <h1>Welcome to Web Scraping</h1>
    <p class="description">This is an example paragraph.</p>
    <a href="https://example.com">Visit Example</a>
  </body>
</html>
"""
# BeautifulSoup 객체 생성
soup = BeautifulSoup(html_doc, 'html.parser')

# 태그 탐색
print(soup.title.string)
print(soup.h1.string)
print(soup.a['href'])
```

```
Example Page
Welcome to Web Scraping
https://example.com
```

- BeautifulSoup을 활용한 웹 스크래핑

  - 주요 기능

    1) **특정 태그 찾기**

       태그 이름으로 요소를 쉽게 찾을 수 있음

```
# 첫 번째 <p> 태그 찾기
print(soup.p)
print(soup.p['class'])
```

```
<p class="description">This is an example paragraph.</p>
['description’]
```

- BeautifulSoup을 활용한 웹 스크래핑

  - 주요 기능

    2) **여러 태그 찾기**

    find_all()을 사용하면 특정 태그를 모두 찾을 수 있음

```
# 모든 <a> 태그 찾기
links = soup.find_all('a')
for link in links:
    print(link['href'])
```

```
https://example.com
```

- BeautifulSoup을 활용한 웹 스크래핑

  - 주요 기능

    3) CSS 선택자 활용

    select()를 사용해

    CSS 선택자를 기반으로 요소를 찾음

    - select는 CSS 선택자를 사용하므로 복잡한 선택 조건을 간단하게 표현
    - find나 find_all 보다 가독성이 좋고 직관적

```
# 클래스가 "description"인 <p> 태그 찾기
description = soup.select_one('.description')
print(description.text)
```

```
This is an example paragraph.
```

BeautifulSoup을 활용한 웹 스크래핑

- BeautifulSoup을 활용한 웹 스크래핑

  – BeautifulSoup으로 웹 페이지 스크래핑

```
import requests
from bs4 import BeautifulSoup

# URL 요청
url = "https://news.ycombinator.com/"
response = requests.get(url)

# BeautifulSoup 객체 생성
soup = BeautifulSoup(response.text, 'html.parser')

# 제목과 링크 추출
for item in soup.find_all('a', class_='storylink'):
    print(f"Title: {item.text}")
    print(f"Link: {item['href']}")
```

```
--
```

BeautifulSoup을 활용한 웹 스크래핑

- BeautifulSoup을 활용한 웹 스크래핑

```
html_doc = """
<div class="news-item">
    <h2 class="title">Breaking News: AI Takes Over</h2>
    <a href="https://news.com/article123">Read More</a>
</div>
<div class="news-item">
    <h2 class="title">Another News: Python is Awesome</h2>
    <a href="https://news.com/article456">Read More</a>
</div>
"""
# BeautifulSoup 객체 생성
soup = BeautifulSoup(html_doc, 'html.parser')

# 모든 뉴스 아이템 추출
news_items = soup.find_all('div', class_='news-item')

for item in news_items:
    title = item.find('h2', class_='title').text
    link = item.find('a')['href']
    print(f"Title: {title}")
    print(f"Link: {link}")
```

```
Title: Breaking News: AI Takes Over
Link: https://news.com/article123
Title: Another News: Python is Awesome
Link: https://news.com/article456
```

- BeautifulSoup을 활용한 웹 스크래핑

  - 주요 기능

    3) CSS 선택자 활용

    select()를 사용해

    CSS 선택자를 기반으로 요소를 찾음

```
# 클래스가 "description"인 <p> 태그 찾기
description = soup.select_one('.description')
print(description.text)
```

```
This is an example paragraph.
```

BeautifulSoup을 활용한 웹 스크래핑

- BeautifulSoup을 활용한 웹 스크래핑

```
from bs4 import BeautifulSoup

html_doc = """
<div class="news-item">
    <h2 class="title">Breaking News: AI Takes Over</h2>
    <a href="https://news.com/article123">Read More</a>
</div>
<div class="news-item">
    <h2 class="title">Another News: Python is Awesome</h2>
    <a href="https://news.com/article456">Read More</a>
</div>
"""
# BeautifulSoup 객체 생성
soup = BeautifulSoup(html_doc, 'html.parser')
# CSS 선택자로 모든 .news-item 선택
news_items = soup.select('.news-item')
for item in news_items:
    title = item.select_one('.title').text  # .title 클래스의 텍스트
    link = item.select_one('a')['href']     # 첫 번째 <a> 태그의 href 속성
    print(f"Title: {title}")
    print(f"Link: {link}")
```

```
Title: Breaking News: AI Takes Over
Link: https://news.com/article123
Title: Another News: Python is Awesome
Link: https://news.com/article456
```

first coding

동적 웹 페이지 데이터 수집

Selenium을 활용한 동적 데이터 수집

브라우저 자동화 및 요소 조작

Selenium을 활용한 동적 데이터 수집

- Selenium이란?

  - Selenium은 웹 브라우저를 자동으로 제어할 수 있게 해주는 라이브러리
  - 주로 웹 애플리케이션 테스트에 사용
  - 동적 웹 페이지에서 데이터를 수집하는 데에도 활용
  - Python을 포함한 여러 프로그래밍 언어에서 사용
  - 실제 브라우저를 통해 페이지를 로드하고 JavaScript 실행 결과까지 확인할 수 있음

Selenium을 활용한 동적 데이터 수집

- 왜 Selenium이 필요한가?

  - 정적 웹 페이지는 HTML 소스 코드만 파싱하면 데이터를 쉽게 가져올 수 있음
  - 동적 웹 페이지는 JavaScript로 콘텐츠를 생성
    → 일반적인 웹 스크래핑 도구(BeautifulSoup 등)로는 데이터 수집이 어려움
  - Selenium은 브라우저를 직접 제어하여 JavaScript가 로드된 이후의 데이터를 가져올 수 있음

Selenium을 활용한 동적 데이터 수집

- Selenium의 기본 구성 요소

  - WebDriver
    - 브라우저를 제어하는 핵심 도구
    - Chrome, Firefox, Edge 등 다양한 브라우저를 지원

      예: ChromeDriver는 Google Chrome을 제어
  - Browser
    - WebDriver가 제어할 브라우저
    - Selenium을 통해 Chrome 브라우저에서 자동으로 페이지를 열고 데이터를 수집할 수 있음
  - Python 라이브러리
    - Selenium은 Python의 라이브러리로 설치 및 실행

Selenium을 활용한 동적 데이터 수집

- Selenium 설치 및 기본 사용법

  - 라이브러리 설치

    pip install selenium

  - WebDriver 다운로드
    - 브라우저에 맞는 WebDriver를 공식 사이트에서 다운로드

      → Chrome의 경우, ChromeDriver를 사용

      https://developer.chrome.com/docs/chromedriver/downloads?hl=ko

      https://googlechromelabs.github.io/chrome-for-testing/#stable

      chrome://settings/help

Selenium을 활용한 동적 데이터 수집

- Selenium을 활용한 브라우저 자동화 및 요소 조작

  - 브라우저 자동화
    - 사람이 직접 브라우저에서 수행하는 작업(페이지 열기, 버튼 클릭, 폼 입력 등)을 Selenium WebDriver를 사용해 프로그램적으로 실행하는 것
    - 주요 기능
      - 브라우저 열기/닫기
      - 특정 URL로 이동
      - 페이지 로딩 상태 확인
      - 브라우저 창 크기 조정 및 탭 제어
      - JavaScript 실행

```
from selenium import webdriver

# Chrome 브라우저 실행
driver = webdriver.Chrome()

# 특정 URL 열기
driver.get('https://www.google.com')

# 브라우저 닫기
driver.quit()
```

Selenium을 활용한 동적 데이터 수집

- Selenium을 활용한 브라우저 자동화 및 요소 조작

  - **요소 조작**
    - 웹 페이지에서 특정 요소(예: 버튼, 텍스트 입력 창, 링크 등)를 찾아 사용자가 수행할 동작(클릭, 텍스트 입력 등)을 자동화하는 것을 의미
    - 요소를 찾는 방법
      - **웹 요소를 식별 주요 메서드**

        find_element: 한 개의 요소를 찾음.

        find_elements: 여러 요소를 찾음.

- **주요 Locator (위치 지정 방법)**

  By.ID: 요소의 고유 ID로 찾기

  By.NAME: 요소의 이름 속성으로 찾기

  By.CLASS_NAME: 클래스 이름으로 찾기

  By.TAG_NAME: HTML 태그 이름으로 찾기

  By.XPATH: XPath로 찾기

  By.CSS_SELECTOR: CSS 선택자로 찾기

Selenium을 활용한 동적 데이터 수집

- 브라우저 자동화와 요소 조작 통합

```
--
```

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
import time

# WebDriver 실행
driver = webdriver.Chrome()
# Google 홈페이지 열기
driver.get('https://www.google.com')
# 검색창 찾기
search_box = driver.find_element(By.NAME, 'q')
# 검색어 입력 및 Enter
search_box.send_keys('Selenium tutorial')
search_box.send_keys(Keys.RETURN)
# 결과 로드 대기
time.sleep(3)
# 검색 결과 출력
results = driver.find_elements(By.XPATH, '//h3')
for index, result in enumerate(results[:5]):  # 상위 5개 결과만 출력
    print(f"{index + 1}. {result.text}")
# 브라우저 닫기
driver.quit()
```

first coding

API를 활용한 데이터 수집

REST API의 이해

공공 데이터 포털 또는 오픈 API 활용

REST API의 이해

- REST API란 무엇인가?

  - REST (Representational State Transfer)
    - 웹 서비스와 클라이언트 간의 통신을 설계하는 아키텍처 스타일
    - REST API는 REST 원칙을 따르는 API(Application Programming Interface)
    - 클라이언트(사용자)와 서버(서비스 제공자) 간 데이터(JSON 또는 XML)를 주고받는 데 사용

REST API의 이해

- **REST API란 무엇인가?**

  - REST API의 구조

    - **URL 구조**

      REST API의 자원 접근은 URL로 구성 → 기본 형식: https://baseurl/resource/{id}

      모든 사용자의 데이터 : https://api.example.com/users

      특정 사용자 데이터 : https://api.example.com/users/123

    - **HTTP 메서드**

      - 서버에 요청을 보낼 때 HTTP 메서드와 URL을 조합해 사용

메서드	동작	설명
GET	데이터 조회	서버에서 데이터를 가져옵니다.
POST	데이터 생성	서버에 새 데이터를 추가합니다.
PUT	데이터 수정	기존 데이터를 업데이트합니다.
DELETE	데이터 삭제	서버에서 데이터를 삭제합니다.

REST API의 이해

- REST API란 무엇인가?

  - REST API의 구조

    - Headers와 Body

      - **Headers**: 요청 또는 응답의 부가정보를 담음. 예: Content-Type, Authorization.
      - **Body**: 요청 시 보내는 데이터. 주로 JSON 형식 사용.

```
POST /users HTTP/1.1
Host: api.example.com
Content-Type: application/json
Authorization: Bearer token123

{
  "name": "king",
  "email": "john.doe@example.com"
}
```

공공 데이터 포털 또는 오픈 API 활용

- **공공 데이터 포털과 오픈 API**

  - **공공 데이터 포털**
    - 정부나 공공기관이 제공하는 데이터를 모아둔 플랫폼
    - 대한민국의 대표적인 공공 데이터 포털: **공공데이터포털(data.go.kr)**
    - **지방 행정인허가데이터개방: LOCALDATA (www.localdata.go.kr)**
    - 다양한 분야의 데이터를 무료로 제공하며, REST API 형식으로 데이터에 접근
    - 주요 데이터
      - 교통 정보 (버스, 지하철 실시간 위치)
      - 기상 정보 (날씨 예보)
      - 인구 통계 (연령별, 지역별 인구)

공공 데이터 포털 또는 오픈 API 활용

- 공공 데이터 포털과 오픈 API

  - 오픈 API
    - 기업이나 기관이 데이터를 제공하기 위해 공개한 API
    - 주요 데이터
      - 네이버 검색, 카카오 검색
      - 구글 지도 API (위치 데이터 제공)
      - 트위터 API (트윗 데이터 제공)
      - NASA API (우주 관련 데이터 제공)

공공 데이터 포털 또는 오픈 API 활용

- **공공 데이터 포털과 오픈 API 활용 데이터 수집**

① 데이터 탐색 및 API 키 발급공공 데이터 포털에서 원하는 데이터 검색

② API 사용 신청을 통해 인증키(API Key)를 발급

→ 이 키는 사용자가 인증된 요청을 보내기 위해 필요

③ API를 사용해 데이터 수집

→ Python의 requests 라이브러리로 API에 요청을 보내 데이터를 수집

→ 데이터는 JSON 형식으로 반환되며, 이를 처리하여 분석에 적합한 형식으로 변환

④ 데이터 전처리

→ 결측치 처리, 중복 제거, 데이터 형식 변환 등을 통해 데이터를 정리

⑤ 데이터 분석

→ Pandas와 Matplotlib 등을 사용해 데이터 시각화 및 통계 분석을 수행

⑥ 머신러닝/딥러닝 모델 학습

→ 데이터에서 유의미한 특징(feature)을 추출하여 머신러닝 및 딥러닝 모델을 훈련합니다.

공공 데이터 포털 또는 오픈 API 활용

- 공공 데이터 포털과 오픈 API 활용 데이터 수집

  – 버스 실시간 위치 데이터 수집 및 분석

    - 목표

    : 특정 버스의 위치를 실시간으로 추적하여 혼잡도 예측

    - 데이터 출처

    : 공공데이터포털의 "버스 실시간 위치 정보 API"

```
import requests
import pandas as pd

# API 키 및 요청 URL
API_KEY = 'YOUR_API_KEY'  # 발급받은 API 키
BASE_URL =
'http://apis.data.go.kr/6260000/BusLocationService/getBusLoca
tionList'
# 요청 매개변수 설정
params = {
    'serviceKey': API_KEY,
    'routeId': '100100',  # 버스 노선 ID (예: 100번 버스)
    '_type': 'json'       # 응답 형식: JSON
}
# API 요청
response = requests.get(BASE_URL, params=params)
data = response.json()  # JSON 데이터 반환
# 데이터 확인
print(data)
# 데이터를 Pandas DataFrame으로 변환
bus_data = pd.DataFrame(data['response']['body']['items'])
print(bus_data)
```

first coding

데이터 저장 및 관리

CSV, Excel, JSON 파일에 데이터 저장

SQLite를 활용한 간단한 데이터베이스 관리

CSV, Excel, JSON 파일에 데이터 저장

- CSV 파일
  - CSV (Comma-Separated Values)
    - 데이터를 쉼표로 구분하여 저장하는 텍스트 기반의 파일 형식.
    - 간단한 데이터 저장과 교환에 적합하며, 대부분의 데이터 분석 도구와 호환
    - Python에서 CSV 파일 저장

```
import csv

# 데이터 준비
data = [
    ['Name', 'Age', 'City'],
    ['Alice', 30, 'New York'],
    ['Bob', 25, 'Los Angeles'],
    ['Charlie', 35, 'Chicago']
]

# CSV 파일에 저장
with open('data.csv', mode='w', newline='',
encoding='utf-8') as file:
    writer = csv.writer(file)
    writer.writerows(data)

print("CSV 파일이 저장되었습니다!")
```

CSV, Excel, JSON 파일에 데이터 저장

- CSV 파일

  - CSV (Comma-Separated Values)
    - CSV 파일 읽기

      pandas 라이브러리를 주로 사용

      - **pd.read_csv('파일경로')**

        : CSV 파일을 읽어 데이터프레임 형태로 반환

      - **data.head()**

        : 상위 몇 개의 데이터를 미리보기.

    - CSV 파일은 단순 텍스트이므로, open()을 사용

```
import pandas as pd

# CSV 파일 읽기
data = pd.read_csv('data.csv')
print(data)
```

```
# open()을 사용한 CSV 파일 읽기
with open('data.csv', 'r') as file:
    for line in file:
        print(line.strip())  # 줄바꿈 제거 후 출력
```

CSV, Excel, JSON 파일에 데이터 저장

- Excel 파일

  - Excel
    - 데이터 시각화와 분석에 강력
    - 표 형태의 데이터를 저장할 수 있음
    - openpyxl 또는 pandas 라이브러리를 주로 사용
    - Excel 파일 쓰기

```
import pandas as pd

# 데이터 준비
data = {
    'Name': ['Alice', 'Bob', 'Charlie'],
    'Age': [30, 25, 35],
    'City': ['New York', 'Los Angeles', 'Chicago']
}

# DataFrame 생성
df = pd.DataFrame(data)

# Excel 파일로 저장
df.to_excel('data.xlsx', index=False)

print("Excel 파일이 저장되었습니다!")
```

CSV, Excel, JSON 파일에 데이터 저장

- Excel 파일

  - Excel
    - Excel 파일 읽기

      pd.read_excel('파일경로'): 엑셀 파일 읽기.

```
data = pd.read_excel('data.xlsx')
print(data)
```

CSV, Excel, JSON 파일에 데이터 저장

- JSON 파일

  - JSON (JavaScript Object Notation)
    - 키-값 쌍으로 데이터를 저장
    - API와 데이터 교환에서 주로 사용
    - 계층적 데이터 구조를 표현할 수 있어 XML의 대안으로 자주 사용
    - Python에서 JSON 파일 저장
      - json.dump()
        : 데이터를 JSON 형식으로 저장.
      - indent=4를 사용하여 사람이 읽기 쉽게 데이터를 정렬

```
import json

# 데이터 준비
data = {
    'people': [
        {'name': 'Alice', 'age': 30, 'city': 'New York'},
        {'name': 'Bob', 'age': 25, 'city': 'Los Angeles'},
        {'name': 'Charlie', 'age': 35, 'city': 'Chicago'}
    ]
}

# JSON 파일로 저장
with open('data.json', mode='w', encoding='utf-8') as
file:
    json.dump(data, file, indent=4)

print("JSON 파일이 저장되었습니다!")
```

CSV, Excel, JSON 파일에 데이터 저장

- JSON 파일

  - JSON (JavaScript Object Notation)
    - Python에서 JSON 파일 읽기
      - json.load()
        : JSON 파일을 파이썬의 딕셔너리로 변환.

```
import json

# JSON 파일 읽기
with open('example.json', 'r') as file:
    data = json.load(file)

print(data)
```

CSV, Excel, JSON 파일에 데이터 저장

- 각 형식의 비교

형식	장점	단점
CSV	간단하고 가볍다. 대부분의 도구와 호환 가능.	계층적 데이터 저장 불가.
Excel	시각화와 다중 시트 지원. 구조화된 데이터 관리.	무겁고 대규모 데이터 교환에 적합하지 않음.
JSON	계층적 데이터 저장 가능. API 데이터 교환에 적합.	사람이 직접 읽기 어렵고 편집 도구가 필요함.

- 활용 팁
  - CSV: 대량 데이터 저장 및 분석 초기 단계에 적합
  - Excel: 데이터 요약, 보고서 작성, 분석 작업에 유용
  - JSON: 웹 및 모바일 애플리케이션과의 데이터 교환에 적합

SQLite를 활용한 간단한 데이터베이스 관리

- SQLite
  - 가볍고 파일 기반의 관계형 데이터베이스 관리 시스템(RDBMS)
  - 서버 설치가 필요 없으며, Python 내장 모듈로 쉽게 사용할 수 있음
  - 데이터 저장, 조회, 관리 등을 SQLite를 활용

SQLite를 활용한 간단한 데이터베이스 관리

- SQLite

① SQLite와 Python 연동

- Python의 sqlite3 모듈을 사용하면
  SQLite 데이터베이스를 쉽게 사용할 수 있음
- 데이터베이스 연결
  - example.db
    SQLite 데이터베이스 파일 이름
    파일이 없으면 새로 생성

```
import sqlite3

# SQLite 데이터베이스 연결 (파일 생성)
connection = sqlite3.connect("example.db")
print("데이터베이스에 연결되었습니다!")

# 연결 닫기
connection.close()
```

SQLite를 활용한 간단한 데이터베이스 관리

- SQLite

② 테이블 생성

- 데이터를 저장하기 위해 먼저 테이블을 생성
- 테이블 생성 코드
  - CREATE TABLE IF NOT EXISTS
    테이블이 없을 경우에만 생성.
  - PRIMARY KEY AUTOINCREMENT
    기본 키로 자동 증가 값 사용.

```
import sqlite3

# 데이터베이스 연결
connection = sqlite3.connect("example.db")
cursor = connection.cursor()

# SQL 명령어로 테이블 생성
cursor.execute("""
CREATE TABLE IF NOT EXISTS users (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    age INTEGER,
    city TEXT
)
""")
print("테이블이 생성되었습니다!")

# 연결 종료
connection.close()
```

SQLite를 활용한 간단한 데이터베이스 관리

- SQLite

③ 데이터 삽입

- Python 코드로 데이터 삽입
  - ?는 값의 자리표시자
    SQL 인젝션 공격을 방지
  - executemany()
    여러 데이터를 한 번에 삽입할 수 있음

```
# 데이터베이스 연결
connection = sqlite3.connect("example.db")
cursor = connection.cursor()

# 데이터 삽입
cursor.execute("INSERT INTO users (name, age, city)
VALUES (?, ?, ?)",
               ("Alice", 30, "New York"))

# 여러 데이터 삽입
data = [
    ("Bob", 25, "Los Angeles"),
    ("Charlie", 35, "Chicago"),
    ("Diana", 28, "Houston")
]
cursor.executemany("INSERT INTO users (name, age, city)
VALUES (?, ?, ?)", data)

# 변경사항 저장
connection.commit()
print("데이터가 삽입되었습니다!")

# 연결 종료
connection.close()
```

SQLite를 활용한 간단한 데이터베이스 관리

- SQLite

④ 데이터 조회

- Python 코드로 데이터 조회
  - fetchall()
    모든 조회 결과를 가져옴
  - fetchone()
    한 행씩 가져옴

```
# 데이터베이스 연결
connection = sqlite3.connect("example.db")
cursor = connection.cursor()

# 데이터 조회
cursor.execute("SELECT * FROM users")
rows = cursor.fetchall()

# 조회된 데이터 출력
for row in rows:
    print(row)

# 연결 종료
connection.close()
```

SQLite를 활용한 간단한 데이터베이스 관리

- SQLite

⑤ 데이터 수정

- Python 코드로 데이터 수정

```
# 데이터베이스 연결
connection = sqlite3.connect("example.db")
cursor = connection.cursor()

# 데이터 수정
cursor.execute("UPDATE users SET age = ? WHERE
name = ?", (31, "Alice"))

# 변경사항 저장
connection.commit()
print("데이터가 수정되었습니다!")

# 연결 종료
connection.close()
```

SQLite를 활용한 간단한 데이터베이스 관리

- SQLite

⑥ 데이터 삭제

- Python 코드로 데이터 삭제

```
# 데이터베이스 연결
connection = sqlite3.connect("example.db")
cursor = connection.cursor()

# 데이터 삭제
cursor.execute("DELETE FROM users WHERE name = ?",
("Bob",))

# 변경사항 저장
connection.commit()
print("데이터가 삭제되었습니다!")

# 연결 종료
connection.close()
```

first coding

데이터 시각화 1 - 기본 시각화

Matplotlib 의 활용

선형 그래프

막대 그래프

히스토그램

Matplotlib 의 활용

- Matplotlib

  - matplotlib은 Python에서 데이터를 시각화하기 위한 가장 기본적인 라이브러리
    - 선형 그래프,
    - 막대 그래프,
    - 히스토그램과 같은 다양한 형태의 시각화를 쉽게 구현

Matplotlib 의 활용

- Matplotlib

  - 선형 그래프 (Line Plot)
    - 선형 그래프는 데이터 점을 선으로 연결하여 값의 변화 추이를 시각화 하는 그래프
    - 주로 시간에 따른 변화, 연속적인 데이터 비교에 사용
    - 그래프 생성 옵션
      - plt.plot: 선형 그래프를 생성하는 함수.
      - marker: 각 데이터 포인트를 나타내는 기호.
      - linestyle: 선 스타일 지정.
      - color: 선 색상.
      - plt.grid: 배경에 격자를 추가하여 가독성 향상.

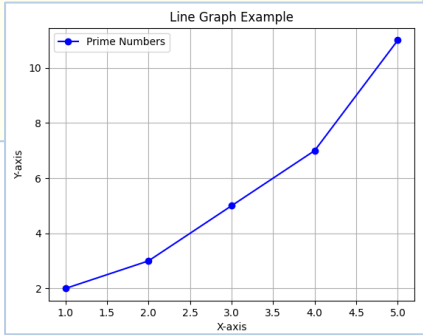
```
import matplotlib.pyplot as plt

# 데이터 준비
x = [1, 2, 3, 4, 5]  # x축 데이터
y = [2, 3, 5, 7, 11]  # y축 데이터

# 그래프 생성
plt.plot(x, y, marker='o', linestyle='-', color='b',
label='Prime Numbers')

# 그래프 꾸미기
plt.title("Line Graph Example")  # 제목
plt.xlabel("X-axis")  # x축 레이블
plt.ylabel("Y-axis")  # y축 레이블
plt.legend()  # 범례 표시

# 그래프 출력
plt.grid(True)  # 격자 추가
plt.show()
```



Matplotlib 의 활용

- Matplotlib

  - 막대 그래프 (Bar Chart)
    - 막대 그래프는 카테고리형 데이터를 시각화
    - 각 막대의 높이 또는 길이가
      해당 데이터 값의 크기를 나타냄
    - 그래프 생성 옵션
      - plt.bar: 막대 그래프를 생성하는 함수.
      - color: 막대 색상 지정.
      - edgecolor: 막대 테두리 색상.

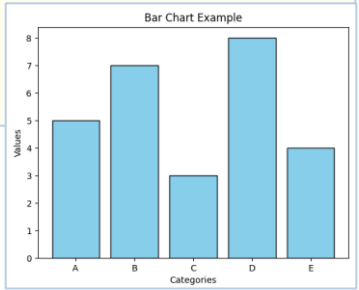
```
import matplotlib.pyplot as plt

# 데이터 준비
categories = ['A', 'B', 'C', 'D', 'E']
values = [5, 7, 3, 8, 4]

# 막대 그래프 생성
plt.bar(categories, values, color='skyblue',
edgecolor='black’)
#plt.barh(categories, values, color='lightgreen',
edgecolor='black')  # 수평 그래프

# 그래프 꾸미기
plt.title("Bar Chart Example")  # 제목
plt.xlabel("Categories")  # x축 레이블
plt.ylabel("Values")  # y축 레이블

# 그래프 출력
plt.show()
```



Matplotlib 의 활용

- Matplotlib

  - 히스토그램 (Histogram)
    - 히스토그램은 데이터를 구간별로 나누어 빈도를 시각화 하는 그래프
    - 주로 데이터 분포를 파악할 때 사용
    - 그래프 생성 옵션
      - plt.hist: 히스토그램을 생성하는 함수
      - bins: 데이터를 나눌 구간 수
      - alpha: 막대 투명도 설정(0.0 ~ 1.0)

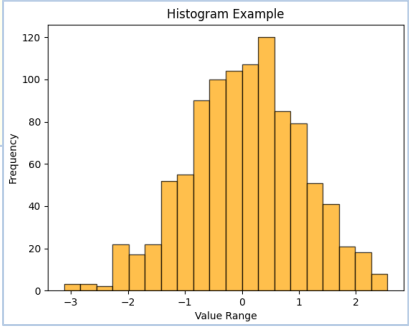
```python
import matplotlib.pyplot as plt
import numpy as np

# 데이터 준비
data = np.random.randn(1000)
# 정규분포를 따르는 난수 1000개 생성

# 히스토그램 생성
plt.hist(data, bins=20, color='orange',
edgecolor='black', alpha=0.7)

# 그래프 꾸미기
plt.title("Histogram Example")  # 제목
plt.xlabel("Value Range")  # x축 레이블
plt.ylabel("Frequency")  # y축 레이블

# 그래프 출력
plt.show()
```



first coding

데이터 시각화 2 - 고급 시각화

Seaborn 의 활용

히트맵

박스플롯

바이올린 플롯

Seaborn 의 활용

- Seaborn

  - Seaborn은 데이터 시각화를 위해 설계된 Python 라이브러리
  - 복잡한 데이터를 쉽게 이해할 수 있도록 다양한 플롯을 제공
    - 히트맵
    - 박스플롯
    - 바이올린 플롯

Seaborn 의 활용

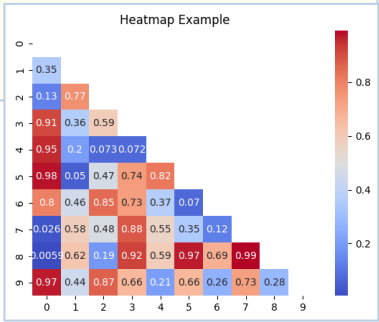
- Seaborn

  - 히트맵 (Heatmap)
    - 히트맵은 데이터의 관계를
      행(row)과 열(column)로 나열하고,
      값의 크기를 색상으로 표현
    - 사용 사례
      - 변수 간의 상관 관계 확인
      - 범주형 데이터의 빈도 시각화
    - 그래프 생성 옵션
      - annot=True: 셀에 숫자를 표시
      - cmap: 색상 팔레트 (예: 'coolwarm', 'viridis')
      - mask: 특정 영역 숨기기 (예: 상관 행렬의 상단 또는 하단)

```python
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# 예제 데이터 생성: 상관관계 행렬
data = np.random.rand(10, 10)
mask = np.triu(np.ones_like(data, dtype=bool))  # 상단 삼각형을 마스킹
sns.heatmap(data, annot=True, mask=mask, cmap='coolwarm')

plt.title("Heatmap Example")
plt.show()
```



Seaborn 의 활용

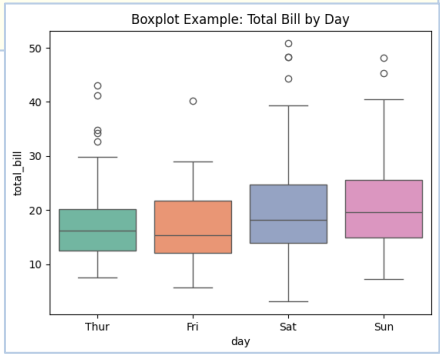
- Seaborn

  - 박스플롯 (Boxplot)
    - 박스플롯은 데이터의 분포를 한눈에 보여주는 플롯
    - 중앙값, 사분위 범위, 이상치를 확인하는 데 사용
    - 사용 사례
      - 데이터의 분포 및 이상치(outliers) 탐지
      - 그룹별 데이터 비교
    - 그래프 생성 옵션
      - 박스(Box): 데이터의 중앙값과 1, 3 사분위수를 나타냄.
      - 수염(Whiskers): 데이터의 최소/최대 값을 표시.
      - 점(Point): 이상치(outliers)를 나타냄.

```python
import seaborn as sns
import matplotlib.pyplot as plt

# 예제 데이터 생성
tips = sns.load_dataset("tips")

sns.boxplot(x="day", y="total_bill", data=tips,
palette="Set2")
plt.title("Boxplot Example: Total Bill by Day")
plt.show()
```



Seaborn 의 활용

- Seaborn

  - 바이올린 플롯 (Violin Plot)
    - 바이올린 플롯은 박스플롯의 기능을 포함하면서 데이터의 분포 밀도를 보여줌
    - 데이터의 대칭성과 분포 패턴을 확인하는 데 유용
    - 사용 사례
      - 그룹 간 분포 차이 시각화
      - 데이터 밀도 확인
    - 그래프 생성 옵션
      - 폭(Width)
        : 데이터 밀도를 나타냄.
        넓을수록 데이터가 해당 값에 많이 분포.
      - 중앙선: 중앙값이나 사분위수.

```python
import seaborn as sns
import matplotlib.pyplot as plt

# 예제 데이터 생성
tips = sns.load_dataset("tips")

sns.violinplot(x="day", y="total_bill", data=tips,
palette="muted", split=True)
plt.title("Violin Plot Example: Total Bill by Day")
plt.show()
```