

생성형AI

Day 4

Python Programming II



목차

1. 고급 자료형
2. Comprehension
3. Logging
4. 정규 표현식
5. Multi-threading & Multi-processing
6. 코딩컨벤션과 테스트
7. REST API
8. Web Framework
9. 실습과제



고급 자료형

리스트(List)

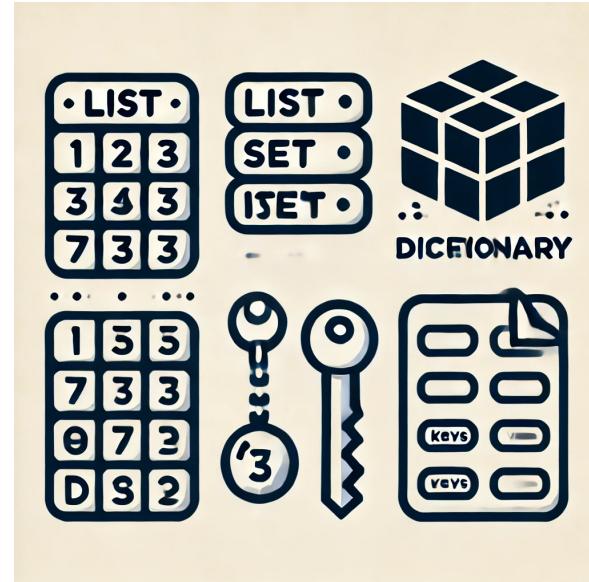
- 순서가 있는 가변 길이의 시퀀스 자료형
- `list = [1, 2, 3, ...]`

세트(Set)

- 순서가 없는 중복되지 않은 요소의 집합
- 중복 제거, 교집합, 합집합, 차집합 등의 집합 연산에 사용
- `set = {'a', 'b', 'c', ...}`

딕셔너리(Dictionary)

- 키-값 쌍으로 이루어진 가변 길이의 자료형
- 빠른 검색, 데이터 맵핑, 데이터 저장 등을 할 때 사용
- `dict = {'a':1, 'b':2, ...}`



고급 자료형

collections 모듈

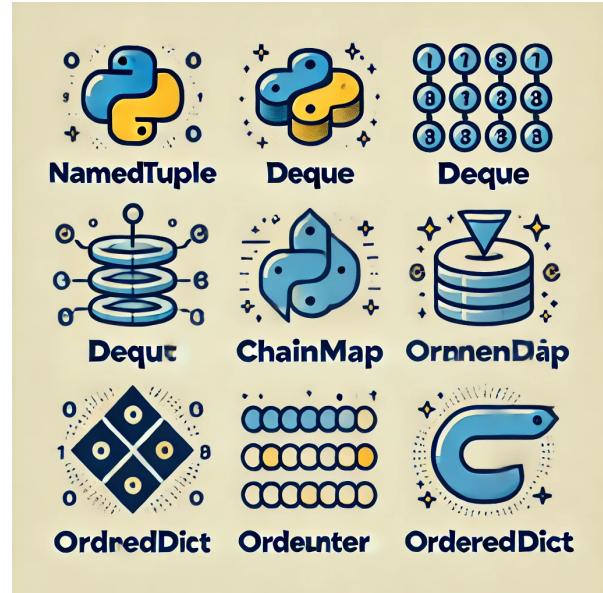
- 파이썬 내장 모듈
- 컨테이너 자료형을 제공하여 기본 자료형을 확장해서 사용 가능

deque (Double-Ended Queue)

- 양쪽 끝에서 빠르게 추가 및 삭제가 가능한 자료형
- `collections.deque`를 사용하여 생성
- 큐와 스택 기능을 동시에 사용해야 할 때 유용

Counter

- 요소의 개수를 셀 때 사용되는 딕셔너리 서브 클래스
- `collections.Counter`를 사용하여 생성
- 데이터의 빈도 분석에 유용



고급 자료형

OrderedDict

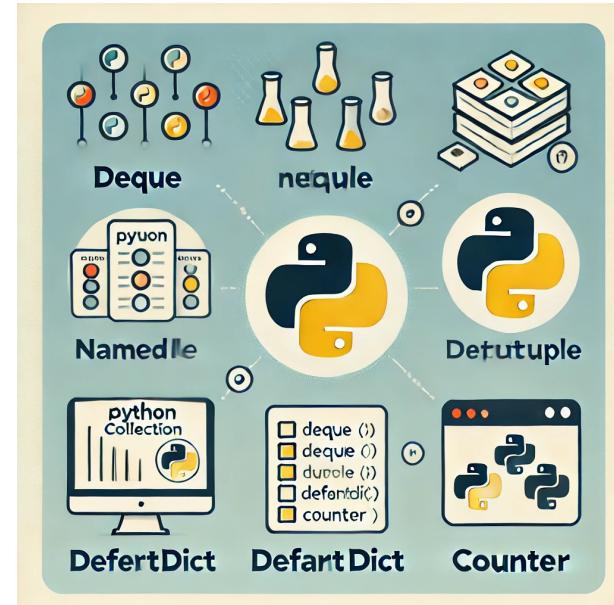
- 순서를 유지하는 딕셔너리
- collections.OrderedDict를 사용하여 생성
- 데이터의 입력 순서가 중요한 경우 사용

defaultdict

- 기본 값을 제공하는 딕셔너리
- collections.defaultdict를 사용하여 생성
- 키가 존재하지 않을 때 기본 값을 제공하여 코드의 간결성과 안전성을 높임

namedtuple

- 필드 이름을 가지는 튜플 서브 클래스
- collections.namedtuple를 사용하여 생성
- 튜플의 가독성을 높이고, 데이터의 구조를 명확히 정의할 때 사용



Comprehension

Comprehension?

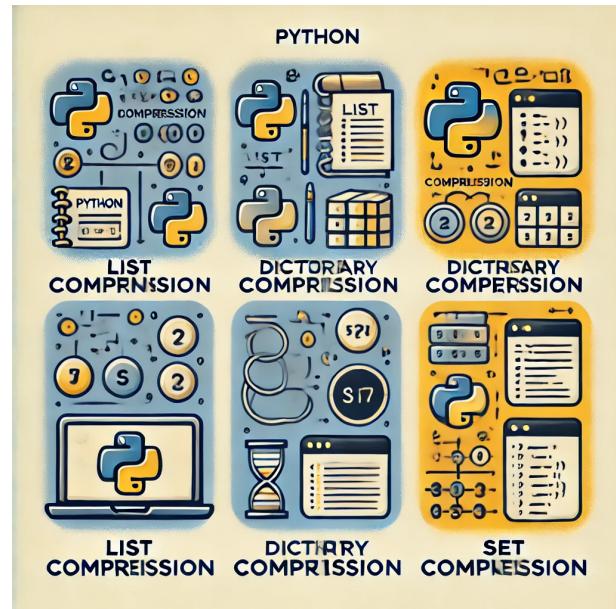
- 기존의 리스트, 집합, 딕셔너리 등을 더 간결하고 효율적으로 생성하는 방법
- 반복문을 대체하여 더 간결한 표현 가능

컴프리헨션 종류

- 리스트 컴프리헨션 (List Comprehension)
- 딕셔너리 컴프리헨션 (Dictionary Comprehension)
- 집합 컴프리헨션 (Set Comprehension)

필요성

- 대규모 데이터를 효율적으로 처리
- 복잡한 반복문을 간결하게 표현



Comprehension

리스트 컴프리헨션

- 기본 구조: [expression for item in iterable]
- 조건부 리스트 컴프리헨션: [expression for item in iterable if condition]

딕셔너리 컴프리헨션

- 기본 구조: {key: value for item in iterable}
- 조건부 딕셔너리 컴프리헨션: {key: value for item in iterable if condition}

집합 컴프리헨션

- 기본 구조: {expression for item in iterable}
- 조건부 집합 컴프리헨션: {expression for item in iterable if condition}

Logging

예외 처리

- 프로그램 실행 중 발생하는 오류를 관리하고 처리하여 예기치 않은 종료를 방지

로깅

- 프로그램 실행 중 발생하는 사건들을 기록하여 디버깅, 오류 추적, 시스템 모니터링 등을 보조

logging 모듈

- 파이썬 표준 라이브러리로 제공되며, 다양한 로그 메시지를 기록할 수 있는 유연한 로깅 시스템을 제공

logging 모듈의 구성 요소

- Logger: 로그 메시지를 기록하는 인터페이스
- Handler: 로그 메시지를 특정 대상(파일, 콘솔 등)으로 전송
- Formatter: 로그 메시지의 형식을 지정
- Level: 로그 메시지의 중요도를 나타냄 (DEBUG, INFO, WARNING, ERROR, CRITICAL)

Logging – 로깅 설정방법

basicConfig

- basicConfig 함수를 사용하여 간단한 로깅 설정을 수행

Logger 객체를 직접 설정

- Logger, Handler, Formatter 객체를 직접 생성하고 설정

설정 파일 사용 (INI, JSON, YAML 등)

- 설정 파일을 사용하여 로깅 설정을 정의하고, 이를 fileConfig, dictConfig 함수를 사용하여 로드

디버깅 및 문제 해결

- 로그 메시지를 통해 코드의 실행 흐름과 상태를 파악할 수 있어 디버깅이 용이

시스템 모니터링

- 시스템의 상태를 실시간으로 모니터링하여 문제 발생 시 빠르게 대응 가능

분석 및 감사

- 로그 기록을 통해 중요한 사건이나 트랜잭션을 추적하고 분석하여 감사와 보고에 활용

정규 표현식

정규표현식

- 문자열에서 특정 패턴을 찾기 위해 사용
- 문자열 검색, 매칭, 치환, 추출 등 다양한 작업을 간단하고 효율적으로 수행
- 데이터 유효성 검사
- 데이터 추출
- 데이터 변환

메타문자	의미
.	임의의 한 문자
^	문자열의 시작
\$	문자열의 끝
*	0회 이상 반복
+	1회 이상 반복
?	0회 또는 1회 반복
{n}	{n}: 정확히 n회 반복
{n,}	{n,}: n회 이상 반복
{n,m}	n회 이상 m회 이하 반복
[abc]	a, b, c 중 하나의 문자
[a-z]	소문자 알파벳 중 하나의 문자
[0-9]	숫자 중 하나의 문자

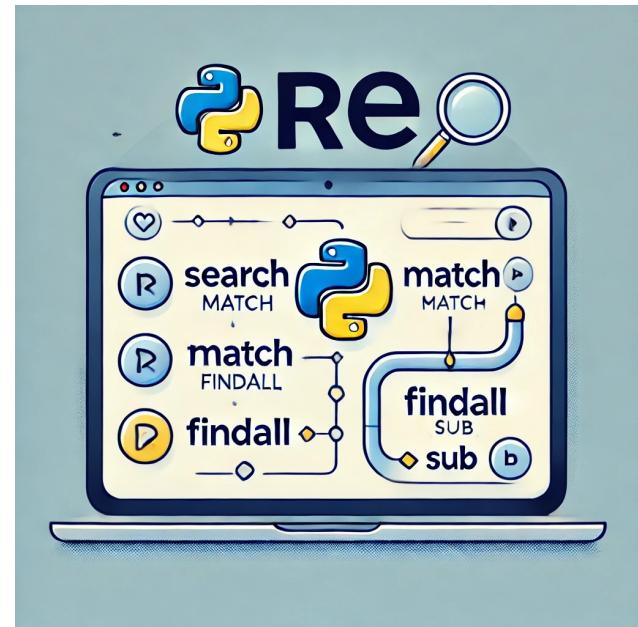
정규 표현식

re 모듈

- 파이썬에서 정규표현식을 처리하기 위해 사용하는 모듈

주요 함수:

- re.match(): 문자열의 시작에서 패턴 매칭
- re.search(): 문자열 전체에서 패턴 매칭
- re.findall(): 매칭되는 모든 패턴을 리스트로 반환
- re.sub(): 매칭되는 패턴을 치환



정규 표현식

그룹핑과 캡처

- 괄호를 사용하여 패턴을 그룹화하고 캡처
- (abc): abc 패턴을 그룹으로 캡처
- (\d{3})-(\d{2})-(\d{4}): 전화번호 형식 캡처

비캡처 그룹: (? : ...) 형식을 사용하여 캡처하지 않는 그룹을 정의.

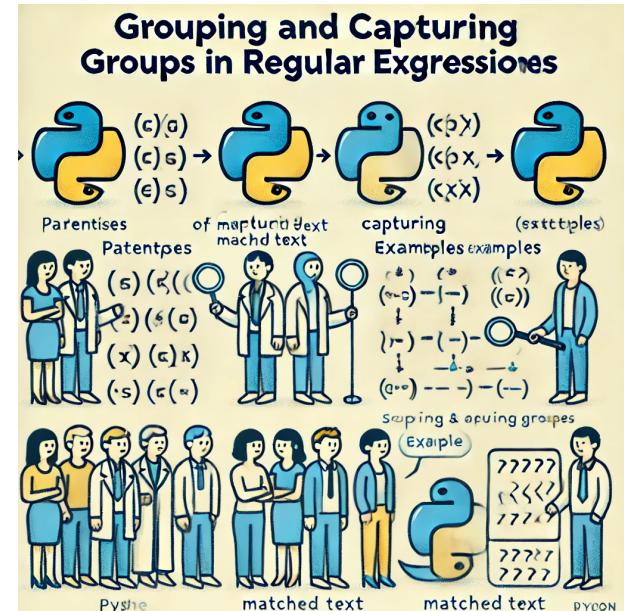
(? : abc): abc 패턴을 그룹으로 묶지만 캡처하지 않음

긍정형 전방탐색: 패턴의 앞부분을 확인하지만 매칭에서 제외 ((?= ...))

\d(?= dollars): 숫자 뒤에 dollars가 오는 경우

부정형 전방탐색: 패턴의 앞부분을 확인하지 않고 매칭에서 제외 ((?! ...))

\d(?!= dollars): 숫자 뒤에 dollars가 오지 않는 경우



Multi-threading & Multi-processing

스레드

- 프로세스 내에서 실행되는 흐름의 단위

프로세스

- 실행중인 프로그램

멀티스레딩

- 하나의 프로세스 내에서 여러 스레드를 생성하여 동시에 실행되는 것처럼 보이게 하는 기술
- 여러 스레드가 동일한 메모리 공간을 공유하며 실행
- 파이썬의 GIL로 인해 진정한 병렬 실행은 아니지만, I/O-bound 작업에서 유용
- 스레드 간 메모리 공유로 데이터 공유가 용이
- CPU-bound 작업에서는 성능 한계
- 스레드 동기화 문제(데드락, 레이스 컨디션 등)

GIL(Global Interpreter Lock)

- 한번에 하나의 기본 스레드만 실행할 수 있도록 스레드 실행을 제어
- 멀티스레딩을 사용하면 I/O 작업(파일 읽기/쓰기, 네트워크 통신 등)을 병렬로 처리하여 프로그램의 응답성을 높임

Multi-threading & Multi-processing

멀티프로세싱

- 여러 개의 프로세스를 생성하여 병렬로 실행하는 기술
- 각 프로세스는 별도의 메모리 공간을 가지며 독립적으로 실행
- GIL의 제약을 받지 않아 병렬 처리가 가능
- CPU-bound 작업을 병렬로 처리하여 성능을 극대화 가능
- 각 프로세스가 독립된 메모리 공간을 가지므로 메모리 충돌 문제 없음
- 프로세스 간 데이터 공유가 어렵고 비용이 높음



Multi-threading & Multi-processing

동기화

- 여러 스레드/프로세스가 공유 자원에 동시에 접근할 때 발생할 수 있는 문제를 방지하기 위한 기법
- 락을 사용하여 데드락을 방지하고, 타임아웃을 설정하여 무한 대기 방지
- 여러 스레드/프로세스가 동시에 데이터베이스나 파일에 접근할 때 데이터 무결성 보호
- 자원 경쟁을 최소화하고 성능 최적화

Lock (락)

- 한 번에 하나의 스레드/프로세스만 자원에 접근할 수 있도록 하는 동기화 기법

Semaphore (세마포어)

- 제한된 수의 스레드/프로세스만 자원에 접근할 수 있도록 하는 동기화 기법

코딩 컨벤션과 테스트

코딩 컨벤션

- 일관된 코드 스타일과 형식을 정의한 규칙 집합
- 코드의 가독성을 높이고, 유지보수를 용이하게 하며, 협업 시 일관성을 유지
- 코드 품질 향상, 디버깅 및 테스트 용이, 팀 간의 효율적 협업

주요 코딩 컨벤션

- PEP 8 (파이썬): 파이썬의 공식 스타일 가이드
 - 들여쓰기: 공백 4칸
 - 라인 길이: 최대 79자
 - 함수 및 변수 이름: 소문자와 밑줄 사용 (예: my_function)
 - 클래스 이름: 단어의 첫 글자를 대문자로 (예: MyClass)
 - 주석: 명확하고 간결하게 작성
 - 공백: 연산자 주변, 함수 인자 사이, 블록 내부 적절히 사용



코딩 컨벤션과 테스트

Linter

- 코드 품질을 검사하고 스타일 가이드를 준수하도록 돕는 도구
- 코드에서 잠재적인 오류, 스타일 위반, 성능 문제 등을 식별

Formatter

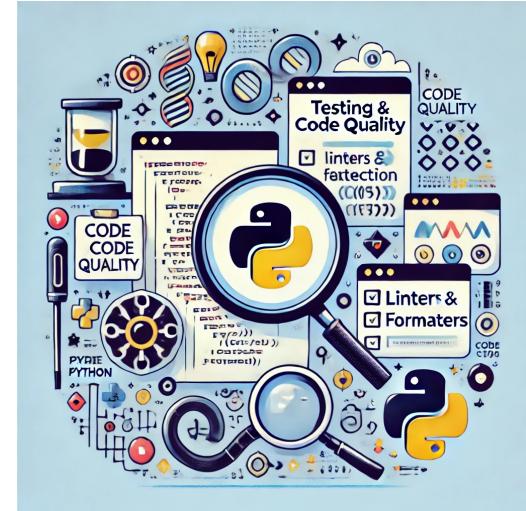
- 코드의 포맷, 스타일을 검사하는 도구

주요 라이브러리

- Black: 파이썬 코드 자동 포매터로, 일관된 스타일을 유지
- Flake8: 파이썬 코드 스타일 검사 도구
- Pylint: 파이썬 코드의 잠재적 버그, 스타일 가이드 위반 등을 검사
- ruff: rust로 작성된 파이썬 린터, 속도가 빠름

Google Python Style Guide

- <https://google.github.io/styleguide/pyguide.html>



코딩 컨벤션과 테스트

단위 테스트

- 소프트웨어의 작은 단위(함수, 메서드)를 테스트하는 기법
- 각 단위가 예상대로 동작하는지 확인하기 위해 테스트 케이스를 작성해서 진행
- 버그 조기 발견
- 코드 품질 향상
- 리팩토링 시 안전성 확보
- python unittest module 사용

TDD (Test Driven Development)

- 테스트 주도 개발(Test Driven Development, TDD)은 테스트를 먼저 작성하고 이를 기반으로 코드를 구현하는 소프트웨어 개발 방법론
- "테스트 작성 -> 코드 구현 -> 리팩토링"의 사이클을 반복하여 소프트웨어를 개발
- 코드의 결함을 조기에 발견하고, 높은 품질의 코드를 빠르게 작성할 수 있습니다.
- 리팩토링 용이
- 테스트를 통해 요구사항을 명확히 정의하고 구현
- 초기 개발 속도 저하
- 테스트 유지보수 비용

코딩 컨벤션과 테스트

테스트 자동화

- 테스트 스크립트를 자동으로 실행하고 결과를 확인하는 기법
- CI/CD 파이프라인에 통합하여 지속적인 테스트와 배포를 자동화
- 테스트 효율성 향상, 인적 오류 감소, 빠른 피드백 제공

코드 테스트 라이브러리/프레임워크

- pytest: 강력하고 유연한 파이썬 테스트 프레임워크
- Jenkins: 오픈 소스 자동화 서버로, CI/CD 파이프라인 구성
- GitHub Actions: GitHub 리포지토리에서 CI/CD 워크플로우를 자동화

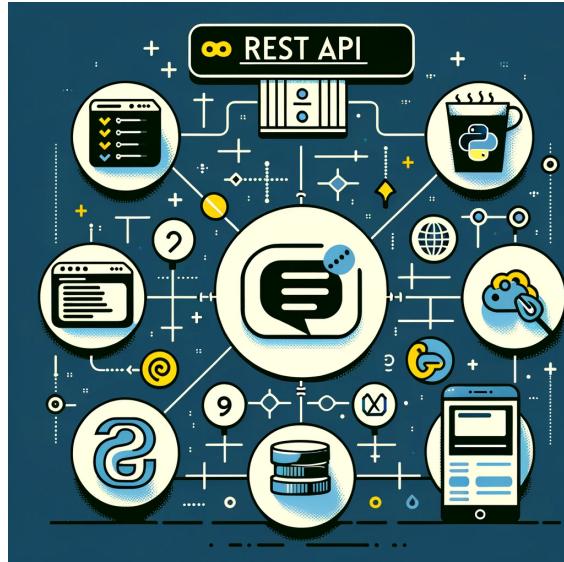
REST API

HTTP 요청

- 클라이언트가 서버에 자원을 요청하는 메시지

주요 HTTP 메서드

- GET: 자원의 조회
- POST: 자원의 생성
- PUT: 자원의 수정
- DELETE: 자원의 삭제



REST API

REST API

- REST(Representational State Transfer) 아키텍처 스타일을 따르는 웹 서비스의 인터페이스
- HTTP 메서드(GET, POST, PUT, DELETE)를 사용하여 자원에 대한 CRUD(Create, Read, Update, Delete) 작업을 수행
- 플랫폼 및 언어에 독립적

REST 원칙

- 클라이언트-서버 구조: 클라이언트와 서버는 서로 독립적으로 동작하며, 클라이언트는 사용자 인터페이스를 담당하고 서버는 데이터 저장 및 처리를 담당
- 무상태성(Stateless): 각 요청은 독립적으로 처리되며, 서버는 요청 간의 상태를 저장하지 않음
- 캐시 가능성(cacheable): 서버 응답은 캐시될 수 있으며, 클라이언트는 캐시된 데이터를 재사용 가능
- 계층화 시스템(Layered system): 서버는 여러 계층으로 구성될 수 있으며, 클라이언트는 중간 서버에 접근해도 서비스의 기능이 동일하게 유지됨
- 통합된 인터페이스(Uniform interface): 자원은 URI로 식별되며, HTTP 메서드를 통해 접근

Web Framework

Web Framework

- 웹 애플리케이션 개발을 쉽게 하고, 표준화된 방법으로 구조화하는 데 도움을 주는 라이브러리와 도구들의 집합
- 웹 프레임워크는 일반적으로 요청과 응답 처리, 템플릿 렌더링, 데이터베이스 연동, 세션 관리 등의 기능을 제공

Python WebFramework

- Django: 빠른 개발과 보안에 중점을 둔 풀스택 프레임워크
- Flask: 경량화된 구조로 필요에 따라 확장 가능한 마이크로 프레임워크
- FastAPI: 빠르고, 현대적이며, 간단한 API 개발을 위한 프레임워크
- Pyramid: 간단하면서도 확장 가능한 웹 프레임워크

FastAPI

- 파이썬으로 작성된 현대적이고 빠른 웹 프레임워크
- ASGI(Asynchronous Server Gateway Interface) 기반으로 비동기 프로그래밍을 지원하며, 데이터 유효성 검사를 자동으로 처리

FastAPI 특징

- 빠른 성능: 비동기 프로그래밍을 통해 높은 성능을 발휘합니다.
- 자동 문서화: OpenAPI와 JSON Schema를 기반으로 자동 문서화를 제공합니다.
- 유효성 검사: Pydantic을 사용하여 데이터 유효성 검사를 자동으로 처리합니다.
- 타입 힌트: 타입 힌트를 사용하여 코드의 가독성을 높이고, 오류를 사전에 방지합니다.



FastAPI

```
from fastapi import FastAPI, HTTPException # FastAPI 및 HTTPException 모듈을 가져옵니다.

# FastAPI 인스턴스를 생성합니다.
app = FastAPI()

# 데이터를 저장할 리스트를 생성합니다.
data_store = []

# '/items/' 경로에 대한 GET 요청을 처리하는 함수입니다.
@app.get("/items/")
def read_items():
    return data_store # data_store 리스트를 JSON 응답으로 반환합니다.

# '/items/' 경로에 대한 POST 요청을 처리하는 함수입니다.
@app.post("/items/")
def add_item(item: dict):
    data_store.append(item) # 요청된 아이템을 data_store 리스트에 추가합니다.
    return item # 추가된 아이템을 JSON 응답으로 반환합니다.
```

FastAPI

```
# '/items/{item_id}' 경로에 대한 PUT 요청을 처리하는 함수입니다.
@app.put("/items/{item_id}")
def update_item(item_id: int, item: dict):
    if item_id >= len(data_store):
        raise HTTPException(status_code=404, detail="Item not found") # 아이템이 없을 경우 404 에러를 반환합니다.
    data_store[item_id] = item # 아이템을 업데이트합니다.
    return item # 업데이트된 아이템을 JSON 응답으로 반환합니다.

# '/items/{item_id}' 경로에 대한 DELETE 요청을 처리하는 함수입니다.
@app.delete("/items/{item_id}")
def delete_item(item_id: int):
    if item_id >= len(data_store):
        raise HTTPException(status_code=404, detail="Item not found") # 아이템이 없을 경우 404 에러를 반환합니다.
    return data_store.pop(item_id) # 아이템을 삭제하고, 삭제된 아이템을 JSON 응답으로 반환합니다.

# 이 코드는 uvicorn을 사용하여 FastAPI 애플리케이션을 실행합니다.
if __name__ == '__main__':
    import uvicorn
    uvicorn.run(app, host="0.0.0.0", port=8000)
```

이론 예제

1. 파이썬 프로그래밍 II 이론 예제 코드

https://colab.research.google.com/drive/1eVytnWkzlrQ6bshU3OSTxYsXx-Prf_Jc?usp=sharing

실습 과제

1. 파이썬 프로그래밍 II 종합 실습

- https://colab.research.google.com/drive/1DSQsvIW5W5RYO2zl_RrdpV06mJ7Eb-gY?usp=sharing

실습 진행