PYCON KOREA 2022

의사 코드 주도 FastAPI 학습하기

FastAPI를 써본 적 없는 개발자의 온보딩과 학습하기

> Do Anything with python!

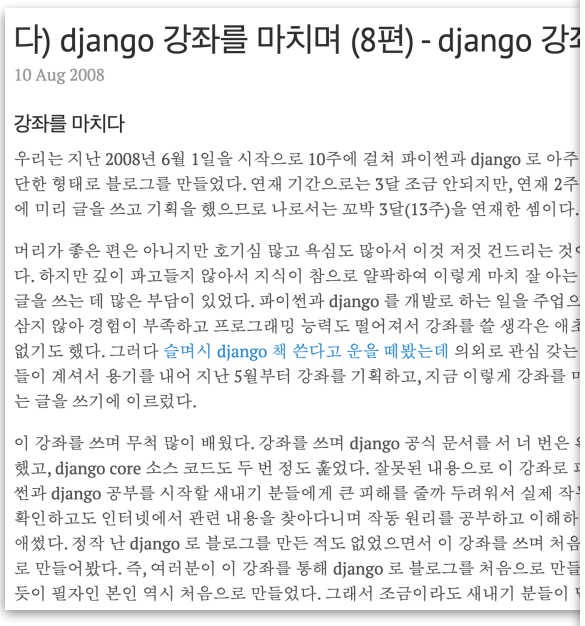
발표자 소개

> Do Anything with python!

- 차경묵 (한날)
  - 학습과 성장, 지속 가능한 개발
  - 심리적 안정감, 다양성, 코칭
  - Django 강좌 (“날로 먹는” 시리즈, 2008, 2014년)
- 경력
  - iamdt™ Integrated Animal Medicine Digital Transformation , 반려동물 의료데이터 스타트업, CTO (현)
  - 게임, 웹, 모바일 분야 등에서 Back-end/Front-end 개발

Django, but

> Do Anything with python!



다) django 강좌를 마치며 (8편) - django 강

10 Aug 2008

강좌를 마치다

우리는 지난 2008년 6월 1일을 시작으로 10주에 걸쳐 파이썬과 django 로 아주
단한 형태로 블로그를 만들었다. 연재 기간으로는 3달 조금 안되지만, 연재 2주
에 미리 글을 쓰고 기획을 했으므로 나로서는 꼬박 3달(13주)을 연재한 셈이다.

머리가 좋은 편은 아니지만 호기심 많고 욕심도 많아서 이것 저것 건드리는 것
다. 하지만 깊이 파고들지 않아서 지식이 참으로 얄팍하여 이렇게 마치 잘 아는
글을 쓰는 데 많은 부담이 있었다. 파이썬과 django 를 개발로 하는 일을 주업으
삼지 않아 경험이 부족하고 프로그래밍 능력도 떨어져서 강좌를 쓸 생각은 애초
없기도 했다. 그러다 슬며시 django 책 쓴다고 운을 떼봤는데 의외로 관심 갖는
들이 계셔서 용기를 내어 지난 5월부터 강좌를 기획하고, 지금 이렇게 강좌를 마
는 글을 쓰기에 이르렀다.

이 강좌를 쓰며 무척 많이 배웠다. 강좌를 쓰며 django 공식 문서를 서 너 번은 읽
했고, django core 소스 코드도 두 번 정도 훑었다. 잘못된 내용으로 이 강좌로 파
썬과 django 공부를 시작할 새내기 분들에게 큰 피해를 줄까 두려워서 실제 작동
확인하고도 인터넷에서 관련 내용을 찾아다니며 작동 원리를 공부하고 이해하
애썼다. 정작 난 django 로 블로그를 만든 적도 없었으면서 이 강좌를 쓰며 처음
로 만들어봤다. 즉, 여러분이 이 강좌를 통해 django 로 블로그를 처음으로 만들
듯이 필자인 본인 역시 처음으로 만들었다. 그래서 조금이라도 새내기 분들이

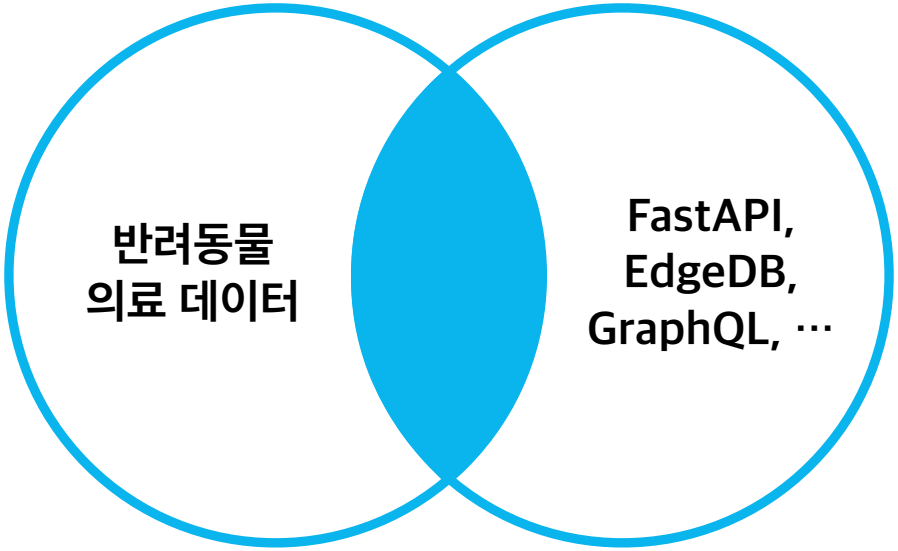
Category: start_with_django_lectures

- Django 1.0에서 Admin 기능 쓰기 - 부록편
  Filed under start_with_django_lectures
  FEB 01 2009
- 다) django 강좌를 마치며 (8편) - django 강좌
  Filed under start_with_django_lectures
  AUG 10 2008
- 이) RSS 기능, 로그인 기능 (7편) - django 강좌
  Filed under start_with_django_lectures
  AUG 04 2008
- 년) 템플릿 작업과 Ajax 작업 (6편) - django 강좌
  Filed under start_with_django_lectures
  JUL 29 2008
- 청) 컨트롤러(뷰) - 댓글 기능 (5편 3/3) - django 강좌
  Filed under start_with_django_lectures
  JUL 20 2008
- 청) 컨트롤러(뷰) - 글 쓰기 기능 (5편 2/3) - django 강좌
  Filed under start_with_django_lectures
  JUL 14 2008
- 청) 컨트롤러(뷰) - 글 목록과 글 보기 기능 (5편 1/3) - django 강좌
  Filed under start_with_django_lectures
  JUL 07 2008
- 미) 파이썬 기초 문법과 데이터 모델링 (4편 2/2) - django 강좌
  Filed under start_with_django_lectures
  JUN 30 2008
- 미) 파이썬 기초 문법과 데이터 모델링 (4편 1/2) - django 강좌
  Filed under start_with_django_lectures
  JUN 22 2008
- 은) 블로그 기획 (3편) - django 강좌
  Filed under start_with_django_lectures
  JUN 16 2008
- 날) Django 설치 (2편) - django 강좌
  Filed under start_with_django_lectures
  JUN 08 2008
- 한) 우리가 만들고자 하는 것과 필요한 기술 (1편) - django 강좌
  Filed under start_with_django_lectures
  JUN 01 2008

Django makes it easier
to build better web apps
more quickly and with less code.

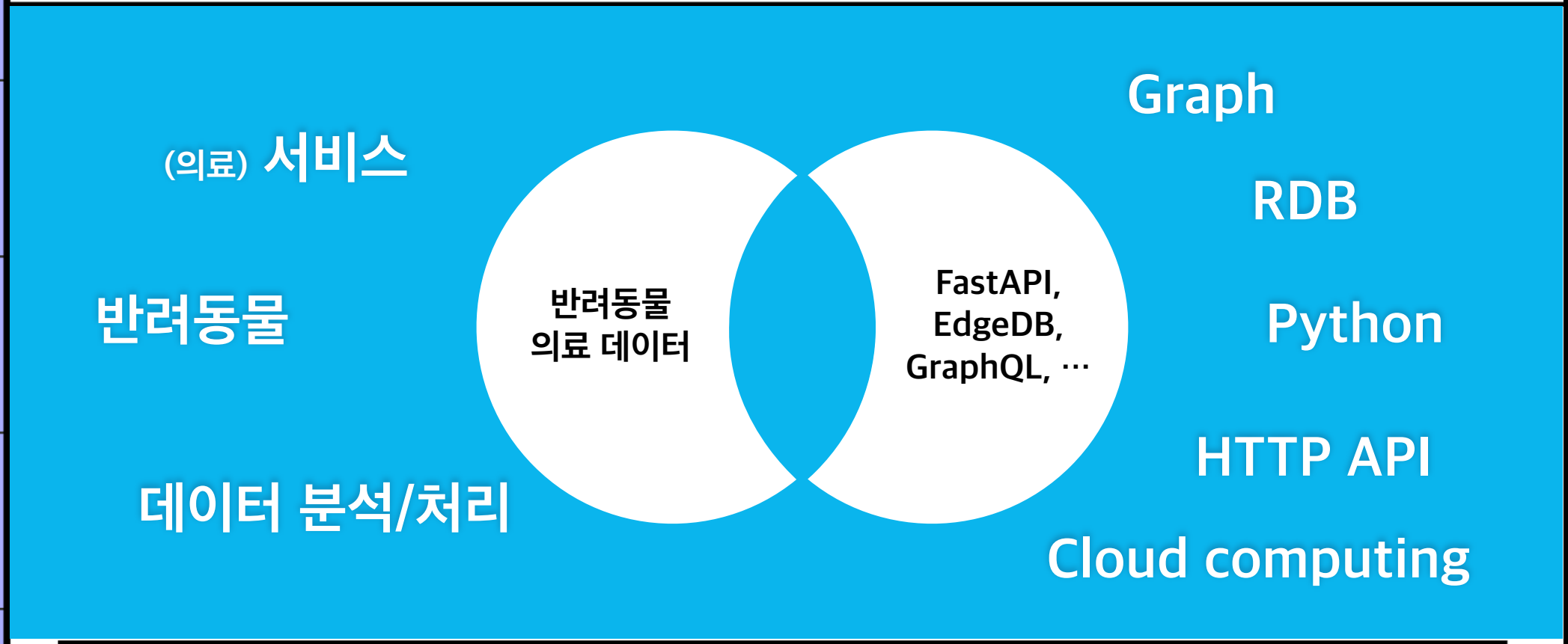
생소 + 생소 + 생소 + …

> Do Anything with python!



근간은 보편적인 도메인과 기술

> Do Anything with python!



새로 합류하는 동료가

우리 팀이 사용하는 도구를

효율적이고 효과적으로 익히려면

어떻게 **온보딩**해야 할까?

우리 팀이

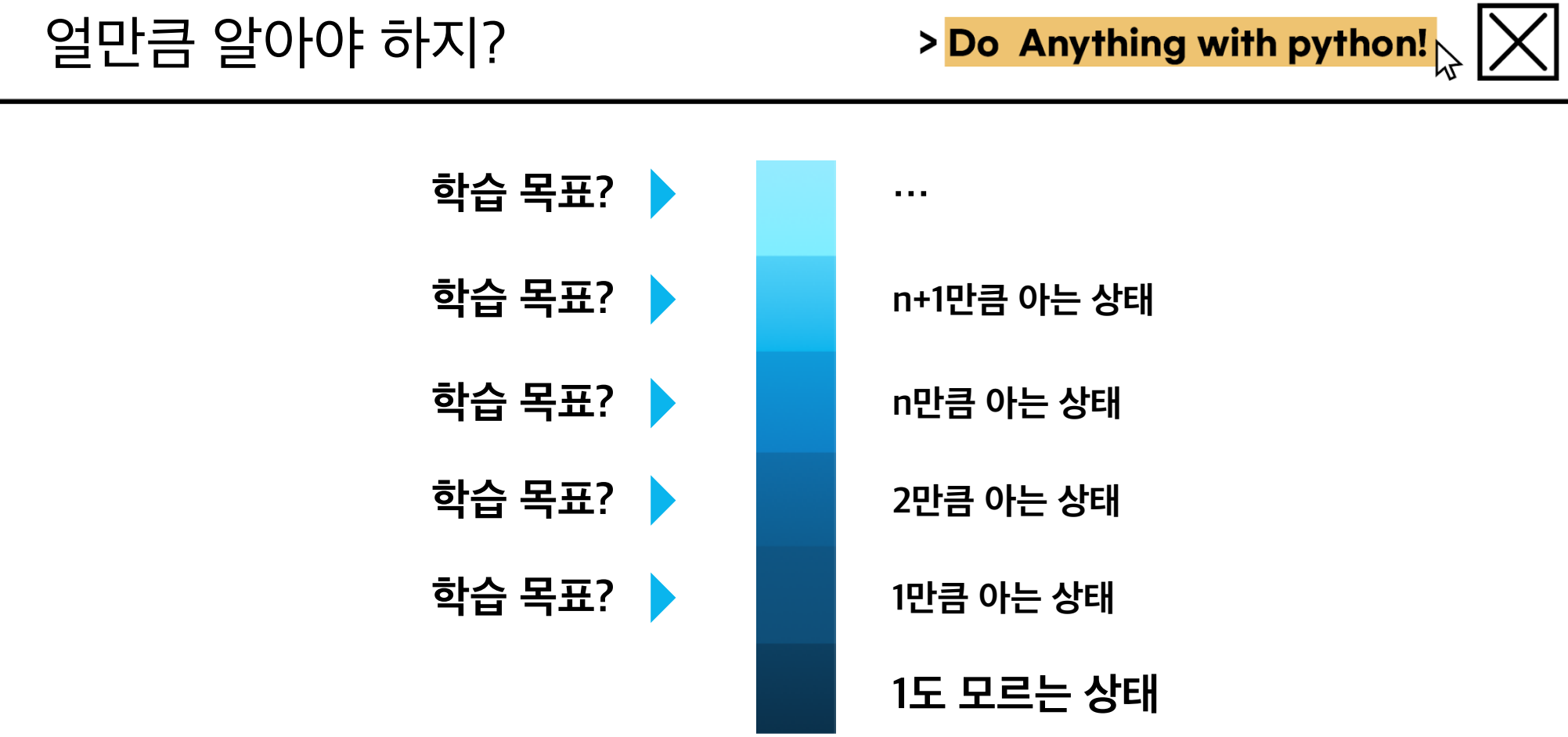
지식과 경험을

효율적이고 효과적으로 익히려면

어떻게 **학습**해야 할까?

얼만큼 알아야 하지?

> Do Anything with python!



목표만큼 아는 상태

> Do Anything with python!

모르는 대상에 대한 학습 목표

- 만들어봐서 잘 아는 것
- 구체화할 수 있을만큼 익숙한 것

목표만큼 아는 상태

n+1만큼 아는 상태

n만큼 아는 상태

2만큼 아는 상태

1만큼 아는 상태

모르는 상태

본진에서부터 개척해가기

> Do Anything with python!

목표만큼 아는 상태

원래+n만큼 알던 걸로 목표를 향해 n+1만큼 더 아는 상태

원래+2만큼 알던 걸로 목표를 향해 n만큼 더 아는 상태

원래+1만큼 알던 걸로 목표를 향해 2만큼 더 아는 상태

원래 알던 걸로 목표를 향해 1만큼 더 아는 상태

모르는 상태

피드백은 아는 영역에서 가까이

> Do Anything with python!

목표만큼 아는 상태

원래+n만큼 알던 것으로 n+1에 대한 피드백을 되먹임 하기

원래+2만큼 알던 것으로 n에 대한 피드백을 되먹임 하기

원래+1만큼 알던 것으로 2에 대한 피드백을 되먹임 하기

원래 알던 것으로 1에 대한 피드백을 되먹임 하기

모르는 상태

Why FastAPI?

> Do Anything with python!

목표한 만큼 아는 상태

- n+1 아는 상태로 전개
- 피드백 범위 좁히기

FastAPI

모르는 상태

발표 주제

> Do Anything with python!

- FastAPI를 학습 교보재로 삼아 학습 방법을 소개, 전파, 공유
- …를 하는 김에 FastAPI도 학습

발표 내용

> Do Anything with python!

1. 만들 기능을 사용자 스토리로 작성
2. 사용자 스토리에 기대하는 동작을 시나리오로 작성
3. 사용자 스토리와 시나리오에 필요한 구현 관련 사항을 추정
4. 추정한 구현 사항에 대해 공식 문서 색인
5. 시나리오를 의사 코드로 작성
6. 공식 문서를 보며 의사 코드를 점진적으로 실 구현 코드로 변환

발표 대상으로 하는 내용

> Do Anything with python!

1. 만들 기능을 **사용자 스토리**로 작성
2. 사용자 스토리에 기대하는 동작을 **시나리오**로 작성
3. 사용자 스토리와 시나리오에 **필요한 구현 관련 사항을 추정**
4. 추정한 구현 사항에 대해 **공식 문서 색인**
5. 시나리오를 **의사 코드**로 작성
6. 공식 문서를 보며 의사 코드를 **점진적으로 실 구현 코드로 변환**

실제로 다루는 내용

> Do Anything with python!

1. 만들 기능을 **사용자 스토리**로 작성
2. 사용자 스토리에 기대하는 동작을 **시나리오**로 작성
3. 사용자 스토리와 시나리오에 **필요한 구현 관련 사항을 추정**
4. 추정한 구현 사항에 대해 **공식 문서 색인**
5. 시나리오를 **의사 코드**로 작성
6. 공식 문서를 보며 의사 코드를 **점진적으로 실 구현 코드로 변환**

구현 환경과 코드 저장소

> Do Anything with python!

- Python (3.10)
- FastAPI (0.83.0)
  - pydantic (1.10.2)
  - SQLAlchemy (1.4)
  - pytest (7.1.3)
- https://bit.ly/hannal-pyconkr2022
  - https://github.com/hannal/hands-on/tree/main/python/pseudo-code-based-learning-fastapi

알고 있는 것 정의하기
(만들 것 정하기)

예약 과정을 가상으로 구성

> Do Anything with python!

- 로그인한 고객은 예약 가능한 세션을 지정해 예약 접수 신청을 한다.

예약 과정을 구체화 예시

> Do Anything with python!

1. 로그인한 고객은 지금 이후 예약 가능한 세션 목록을 본 뒤
2. 예약 가능한 세션을 지정해 예약 접수 신청을 한다.

예약 과정을 구체화 예시

> Do Anything with python!

1. 로그인한 고객은 지금 이후 예약 가능한 세션 목록을 본 뒤
   - 예약 세션은 진료과목 별 시간표의 진료 시간 범위이다
   - 예약은 미래를 대상으로 하므로 현재 기준으로 미래 항목이어야 한다
   - 지정한 월 단위로 예약 세션을 묶어서 목록으로 나열한다
   - 예약 할 수 있는 예약 세션만 나열한다
2. 예약 가능한 세션을 지정해 예약 접수 신청을 한다.
   - 진료과목을 선택하지 않고 일자와 시간대로만 예약 세션으로 정하여 예약 접수한다 (고객이 어떤 진료과목인지 모를 가능성이 크므로)
   - 고객은 자신의 예약 건에 대해 설명을 부연하거나 사진을 첨부할 수 있다.
   - 예약 담당자가 예약 접수 건을 검토하여 예약을 확정 상태로 변경한다

예약 과정을 구체화 예시

> Do Anything with python!

1. 로그인한 고객은 지금 이후 예약 가능한 세션 목록을 본 뒤
    - 예약 세션은 진료과목 별 시간표의 진료 시간 범위이다
    - 예약은 미래를 대상으로 하므로 현재 기준으로 미래 항목이어야 한다
    - 지정한 월 단위로 예약 세션을 묶어서 목록으로 나열한다
    - 예약 할 수 있는 예약 세션만 나열한다
2. 예약 가능한 세션을 지정해 예약 접수 신청을 한다.
    - 진료과목을 선택하지 않고 일자와 시간대로만 예약 세션으로 정하여 예약 접수한다 (고객이 어떤 진료과목인지 모를 가능성이 크므로)
    - 고객은 자신의 예약 건에 대해 설명을 부연하거나 사진을 첨부할 수 있다.
    - 예약 담당자가 예약 접수 건을 검토하여 예약을 확정 상태로 변경한다

사용자 스토리

사용자 스토리 정리 예시

> Do Anything with python!

스토리ID	역할	목적	기능
C-01	로그인한 고객	지금 이후 예약 현황 파악	예약 가능한 세션 목록을 조회
C-02	로그인한 고객	예약 세션을 접수 신청	예약 가능한 세션에 예약 등록
S-01	병원의 예약 담당자	접수된 예약 세션 현황 파악	접수된 예약 세션 목록을 조회
C-03	예약 고객	예약할 진료과를 확정하는 데 필요한 내용을 보완	예약 단계인 자신의 예약 세션에 사진을 첨부/관리
S-02	병원의 예약 담당자	의료 처치 계획을 확정	예약 단계인 예약 세션을 확정 상태로 변경
C-04	예약 고객	예약 담당자가 자신의 예약의 상태를 안내 받기	예약 세션의 상태를 변경하면 예약 고객에게 자동으로 모바일 알림 발송

C-01 스토리에 대한 시나리오

> Do Anything with python!

시나리오ID	주어진 조건	수행	기대하는 결과
C-01-F01	로그인 안 함.	예약 가능한 세션 목록 가져오기	인증되지 않았다는 오류 응답
C-01-S01	로그인 함. 예약 항목 2개.	예약 가능한 세션 목록 가져오기	예약 항목 2개를 목록으로 반환
C-01-S02	로그인 함. 예약된 항목 1개 + 예약 가능 항목 1개.	예약 가능한 세션 목록 가져오기	예약 항목 1개를 목록으로 반환
C-01-S03	로그인 함. 예약된 항목 1개 + 오늘 이후 이번 달 예약 가능 항목 1개 + 다음 달 예약 가능 항목 1개.	이번 달을 지정하여 예약 가능한 세션 목록 가져오기	예약 항목 1개를 목록으로 반환
C-01-F02	로그인 함.	지난 달을 지정하여 예약 가능한 세션 목록 가져오기	잘못된 요청하지 말라는 오류 응답
...			

필요하다고 예상하는 구현 관련 키워드

> Do Anything with python!

- 아는대로
- 되도록 한 줄에 키워드 하나
- 시나리오 별로, 중복되더라도
- 키워드는 되도록 정식/공식 기술 용어로
    - 모르면 정식/공식스러운 기술 용어로
- 시간 많이 걸리고 힘들다면, 💯👍👏

시나리오ID	키워드
C-01-F01	인증(Authentication)
	API 경로(Route)
	오류 응답(HTTP Status code, HTTP Response)
C-01-S01	정상 응답
	JSON 직렬화
C-01-S02	데이터 클래스
	RDB
	영속 데이터 저장소(Repository)
C-01-S03	HTTP Query parameter

키워드에 대한 공식 문서 색인

> Do Anything with python!

- 목적 : 색인하는 것이 목적
- 목표 : 나만의 색인 체계 구축
- FastAPI
  - Starlette + pydantic
  - 두 도구의 공식 문서도 색인하면 더 좋음

시나리오ID	키워드	공식 문서
C-01-F01	인증(Authentication)	Header Parameters
		Security Intro
		Dependencies - First Steps
		Starlette Authentication
		Middleware
		Advanced Middleware
	오류 응답(HTTP Status code, HTTP Response)	Handling Errors
		Starlette Exceptions

나만의 색인 체계 예시

> Do Anything with python!

다단 색인 체계

시나리오ID	키워드	공식문서 주제	연관 소주제
C-01-F01	인증(Authentication)	Dependencies - First Steps	Dependencies - First Steps
			Header Parameters
			Security Intro

공식 문서 내용에 대한 키워드

(network graph model)

시나리오ID	키워드	공식 문서	내 관점의 키워드
C-01-F01	인증(Authentication)	Dependencies - First Steps	의존성 주입
			관심사
			oAuth2
			인지 경계 (인증 계층의 어느 범위만큼 알아야해?)

색인 노트

> Do Anything with python!

- 코딩하며 학습하는 과정에서 알게 된 것을 기록
- 키워드가 같더라도 시나리오에 따라 다른 접근이 필요한 경우 존재

시나리오ID	키워드	공식 문서 주제	연관 소주제	노트
C-01-S01	인증	Dependencies	Dependencies - First Steps	이미 예약된 항목이더라도 자신의 예약 항목은 변경할 수 있으니 로그인한 사용자가 누구인지 알아야 한다.
		...		
C-01-F01	인증	Dependencies	Dependencies in path operation decorators	로그인을 안 한 경우, 목록을 가져오는 기능을 실행 조차 안 한다. 그럼 경로 수행(path operation)에서만 인증 처리하면 될 것 같다.
		...		

정리

> Do Anything with python!

- **사용자 스토리** : 아는 것(만들 대상에 대한 목적 또는 목표) 정의
- **시나리오** : 아는 것을 구체적으로 정리해서 더 잘 알기
- **예상 구현 키워드** : 학습 대상(FastAPI)에 대해 무엇을 학습할지 아는 것을 기준으로 정리
- **공식 문서 색인** : 학습 대상에 대해 모르는 것을 명시/구체화
- **색인 노트** : 앞으로 학습하는 것을 내가 아는 것에 비추어 정리

◀ 여기까지만 진행

C-01, **로그인한 고객**은 **지금 이후 예약 현황 파악**을 하기 위해 **예약 가능한 세션 목록을 조회**한다.

C-01-S01 시나리오, 보편어 의사코드

> Do Anything with python!

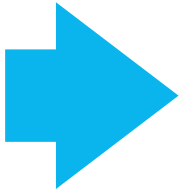
- 주어진 조건
  - 로그인 함.
  - 예약 항목이 2개 있음.
- 수행
  - 예약 가능한 세션 목록 가져오기
- 기대하는 결과
  - 예약 항목 2개를 목록으로 반환

```
def 예약_가능한_세션_목록_가져오기(user):
    return []

def test_예약_가능한_세션_목록_가져오기():
    user = 로그인_했다_치자
    items = [예약항목1, 예약항목2]

    result = 예약_가능한_세션_목록_가져오기(user)

    기대하는_것 items == result
```

C-01-S01 시나리오, Python 의사 코드

> **Do Anything with python!**

```
def 예약_가능한_세션_목록_가져오기(user):
    return []

def test_예약_가능한_세션_목록_가져오기():
    user = 로그인_했다_치자
    items = [예약항목1, 예약항목2]

    result = 예약_가능한_세션_목록_가져오기
(user)

    기대하는_것 items == result
```

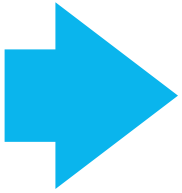


```
def reservations(user):
    return []

def test_reservations():
    # 주어진 조건
    user = "hannal"
    items = [ ··· ]

    # 수행
    result = reservations(user)

    # 기대하는 결과
    assert frozenset(items) == frozenset(result)
```

예약 항목 영속 데이터 저장소 구현

> **Do Anything with python!**

```
ReservationRepository객체
    def create(예약_항목_생성에_필요한_데이터):
        새_예약_항목 = 영속_데이터_생성(예약_항목_생성에_필요한_데이터)
        영속_데이터_저장(새_예약_항목)
        새_예약_항목.id = 고유한_일련번호_값
        return 새_예약_항목

    def request_reservation(고객, 예약_접수에_필요한_데이터):
        예약 항목에 대해 예약 접수 신청을 하고, 변경된 예약 항목을 반환한다.

    def findall():
        return list(저장된_영속 데이터들)

    def find_by_id(고유한_일련번호_값):
        obj = find(영속데이터들, 각_영속_데이터.id == 고유한_일련번호_값)
        if obj가 존재하면:
            return obj
        else:
            return None
```



```
items = []

class ReservationRepository:
    def create(self, payload):
        obj = payload
        obj.id = len(self._items) + 1
        items.append(obj)
        return obj

    def request_reservation(self, customer, payload):
        obj = payload
        obj.customer_id = customer.id
        obj.customer_note = payload.note
        return obj

    def findall(self):
        return items

    def find_by_id(self, id):
        objs = [o for o in items if o.id == id]
        return objs[0] if objs else None
```

ReservationRepository 반영

> Do Anything with python!

- (가능할 경우) 이 단계에서는 테스트를 통과하는 Python 코드 작성
- 테스트 통과가 목적
- 구현 전개와 리팩터링은 당장 고려 사항 아님

```
def reservations(user, repository):
    items = repository.findall()
    return items

def test_reservations():
    repository = ReservationRepository()
    items = [repository.create(item) for item
in …]

    result = reservations(“hannal”, repository)

    assert frozenset(items) == frozenset(result)
```

예약 목록 HTTP API

C-01-S01 확장 시나리오, HTTP API로 접근 > Do Anything with python!

- 주어진 조건
  - 예약 항목이 있음.
  - 로그인 정보가 있는 HTTP Client.
- 수행
  - HTTP Client로 /reservation/reservations URL 을 HTTP GET method 요청
- 기대 결과
  - HTTP Status code 200
  - JSON 형식으로 받은 예약 항목 개수와 이미 저장 된 데이터 개수가 같음

```
from fastapi import status
from fastapi.testclient import TestClient


app = FastAPI()


@pytest.fixture
def reservations_fixture():
    return [ … ]


def test_get_reservation_list(reservations_fixture):
    client = TestClient(app)
    res = client.get("/reservation/reservations", headers={ … })
    assert res.status_code == status.HTTP_200_OK
    data = res.json()
    assert len(data) == len(reservations_fixture)
```

주어진 조건 : 예약 항목이 있음

주어진 조건 : 로그인 정보가 있는 HTTP Client

수행

기대 결과

Testing 색인 노트 : TestClient

> Do Anything with python!

- `fastapi.testclient.TestClient`
  - 실제로는 **starlette**에서 제공하는 TestClient.

```
from starlette.testclient import TestClient as TestClient
```

  - `requests.Session` 클래스를 상속받아 확장한 것이라 **requests**와 거의 동일한 인터페이스.
  - 동기식으로 동작하는 requests이므로 테스팅 함수도 `async`가 아닌 일반 함수.
    - HTTP Authorization header에 인증 토큰을 담는 것으로 인증할 것이므로 HTTP Client의 HTTP Request 보낼 때 `headers` 인자 사용.
- ASGI 인스턴스 객체를 필수 인자로 받음
  - FastAPI 인스턴스 객체 : ASGI 인스턴스 객체 역할 포함
  - FastAPI 인스턴스 객체에 여러 설정을 등록/반영 후 이 객체를 TestClient에 전달하여 사용 (예 : Routing)

공식 문서 활용법

- requests 공식 문서 : HTTP Client의 HTTP 인터페이스에
- starlette 공식 문서 : 테스팅 애플리케이션 서버와 HTTP 연결과 설정
- FastAPI 공식 문서 : FastAPI 테스팅 환경 구성, FastAPI 테스팅 코드 작성

API 경로 연결 (APIRouter)

> Do Anything with python!

```
# controllers.py

from fastapi import APIRouter


router = APIRouter()

@router.get("/reservations")

def reservations():

    ...
```

HTTP GET method에 대해 지정한 URL로 연결(매핑)

```
# main.py

from fastapi import FastAPI

from controllers import router


app = FastAPI()

app.include_router(router, prefix="/reservation")
```

FastAPI 인스턴스 객체에 경로 등록

Routing 색인 노트

> Do Anything with python!

- 애플리케이션 서버에 경로 등록하는 두 가지 방법
  - A : APIRouter 인스턴스 객체 이용
    - starlette routing 공식 문서 : APIRouter 자체에 대한 상세 정보
    - FastAPI 공식 문서 : FastAPI에서 APIRouter 활용 방법 제시
  - B : FastAPI 인스턴스 객체를 직접 이용
    - 간단하지만, 불편.
- Routing 인자는 A, B 동일

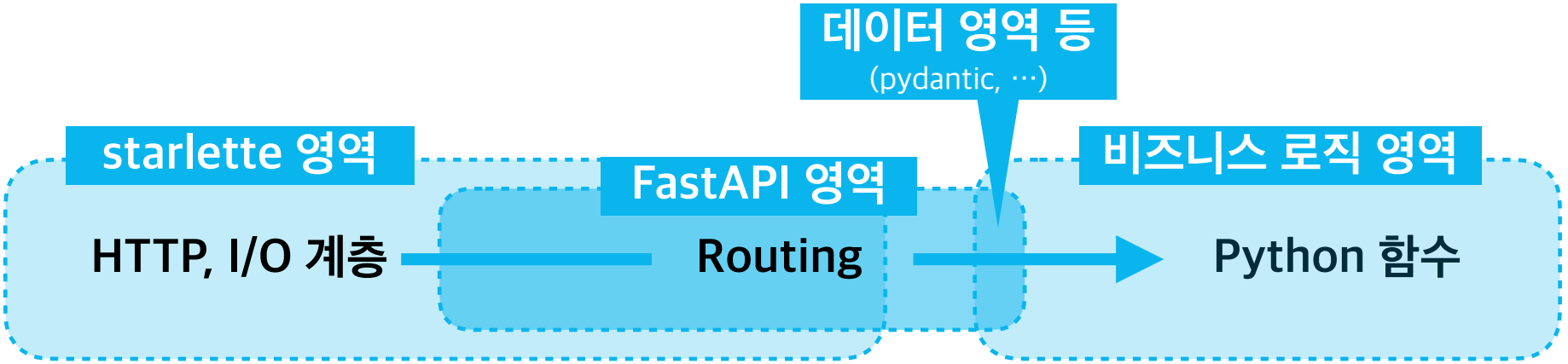
순환 참조 오류 발생

```
# app.py
from fastapi import FastAPI
from . import controllers

app = FastAPI()

# controllers.py
from .app import app

@app.get("/reservations")
def reservations():
    ...
```

main.py으로 순환 참조 회피

```
# app.py
from fastapi import FastAPI

app = FastAPI()

# controllers.py
from .app import app


# main.py
import uvicorn
from .app import app
from . import controllers

if __name__ == "__main__":
    uvicorn.run(app, host="0.0.0.0", port=8000)
```

Routing 색인 노트 : 각 영역의 구조와 역할 > Do Anything with python!



- HTTP (ASGI) 처리
- 비동기(async) 처리

- HTTP 계층과 Python 함수 매핑
- FastAPI가 처리하는 HTTP 계층단 관심사를 Python 함수로 주입, 명시화
- 비동기 작업(task) 단위 의존성 관리
- 데이터 직렬화, 역직렬화

- HTTP 계층에 대한 최소한으로 결합
- 순수 Python으로 비즈니스 로직 의도를 담음

월(month)을 지정해
예약 항목 목록 가져오기

(C-01-S03, C-01-F02)

C-01-F02, 지난 달로 필터링

> Do Anything with python!

- 주어진 조건
  - ~~로그인 함.~~
  - 예약된 항목 1개
  - 오늘 이후 이번 달 예약 가능 항목 1개
  - 다음 달 예약 가능 항목 1개.
- 수행
  - 지난 달을 지정하여 예약 가능한 세션 목록 가져오기
- 기대 결과
  - 잘못된 요청하지 말라는 오류 응답

```
@pytest.mark.asyncio
async def test_reservations():
    repository = ReservationRepository()
    items = [await repository.create(item) for item in …]

    scheduled_date = datetime.date(2020, 1, 1)

    with pytest.raises(ValidationError):
        await reservations(“hannal”, repository, scheduled_date)
```

과거 월 검사하기

> Do Anything with python!

- `fastapi.exceptions.ValidationError` 사용
  - `pydantic.error_wrappers.ValidationError`
- HTTP 계층을 드러내지 않고 Python 영역만으로 구현 충족

```
def validate_scheduled_date(v: datetime.date):
    today = datetime.date.today()
    if v.year < today.year or v.month < today.month:
        raise ValidationError("오늘이거나 이후 일자여야 합니다.")

    if v.year == today.year and v.month == today.month:
        return today
    return v


@router.get("/reservations", response_model=list[schemas.Reservation])
async def reservations(user, repository, scheduled_date: datetime.date):
    validate_scheduled_date(scheduled_date)
    ...
    return items
```

Query 인자를 자동으로 형 변환하여 전달 > Do Anything with python!

- 함수의 인자를 경로 parameter나 Query parameter로 자동으로 연결.
  - type annotation이 있으면 type 변환하며 간단한 validation도 이뤄짐.
- 공식문서
  - Query Parameters
  - Query Parameters and String Validations

```
def validate_scheduled_date(v: datetime.date):
    today = datetime.date.today()
    if v.year < today.year or v.month < today.month:
        raise ValidationError("오늘이거나 이후 일자여야 합니다.")

    if v.year == today.year and v.month == today.month:
        return today
    return v


@router.get("/reservations", response_model=list[schemas.Reservation])
async def reservations(user, repository, scheduled_date: datetime.date):
    validate_scheduled_date(scheduled_date)
    ...
    return items
```

로그인 여부와
로그인한 이용자 정보 가져오기

(C-01-S03)

OAuth2 인증 토큰으로 인증, 테스팅

> Do Anything with python!

- 주어진 조건
  - 로그인 함
  - 로그인 안 함
- 수행
  - 이번 달을 지정하여 예약 가능한 세션 목록 가져오기
- 기대 결과
  - 로그인 한 경우 예약 세션 목록 반환
  - 로그인 안 한 경우 로그인을 해야 한다고 응답

```
@pytest.mark.parametrize(
    “headers, expected_status_code”,
    [
        [None, status.HTTP_401_UNAUTHORIZED],
        [{"Authorization": "Bearer hannal"}, status.HTTP_200_OK],
    ],
)
def test_get_reservation_list_without_auth(headers, expected_status_code):
    client = TestClient(app)
    params = {
        “scheduled_date”: datetime.date.today().isoformat(),
    }

    res = client.get(“/reservation/reservations”, params, headers=headers)
    assert res.status_code == expected_status_code
```

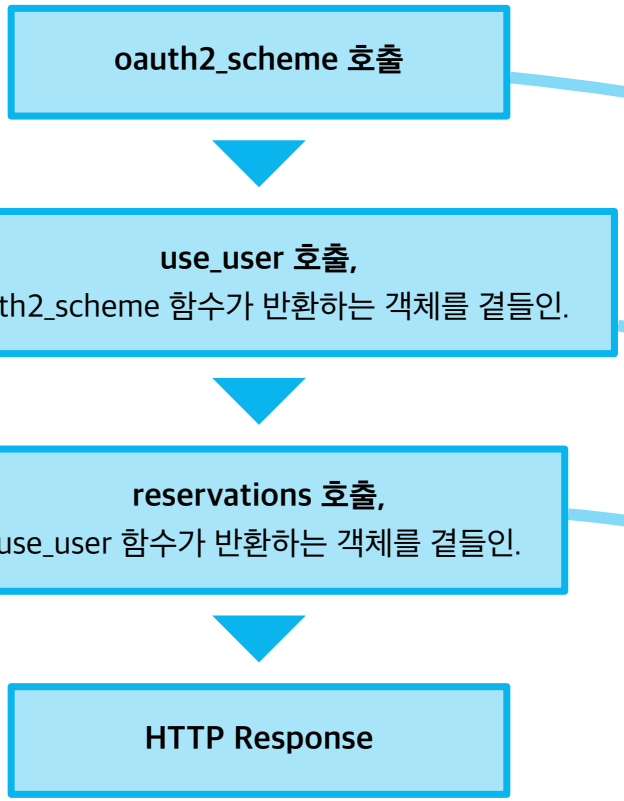
분리한 관심사를 FastAPI가 주입하여 호출하기 > Do Anything with python!

- **Depends**
  - FastAPI가 컨트롤러(함수)를 호출할 때, FastAPI가 컨트롤러에 전달해줄 의존 대상을 지정하는 객체
  - 매우 빈번하게 사용

```
from fastapi import Depends
from fastapi.security import OAuth2PasswordBearer
from fastapi.exceptions import HTTPException

oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")

async def use_user(token: str = Depends(oauth2_scheme)):
    if token != “hannal”:
        raise HTTPException(status_code=status.HTTP_401_UNAUTHORIZED)
    return User(username=token)

@router.get(“/reservations”, …, dependencies=[Depends(use_user)])
async def reservations(…):
    …
```

의존성(dependency) 주입과 동작 흐름

> Do Anything with python!




```python
from fastapi import Depends
from fastapi.security import OAuth2PasswordBearer
from fastapi.exceptions import HTTPException

5
oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")

3
async def use_user(token: str = Depends(oauth2_scheme)):
    if token != “hannal”:
        raise HTTPException(status_code=status.HTTP_401_UNAUTHORIZED)
    return User(username=token)

@router.get(“/reservations”, …, dependencies=[Depends(use_user)])
1
async def reservations(…):
    …
```

의존성을 Controller에 주입하는 두 가지 방법

> Do Anything with python!

- Routing에 의존 대상들을 나열 1
  - 로그인한 이용자이기만 하면 됨.
  - Python 함수로써 reservations는 로그인 여부에 대해 몰라도 됨.
- Python 함수의 인자 기본값으로 Depends 객체 지정 2 3
  - Python 함수로써 reservations의 user 인자는 인자 기본값으로 Depends 객체가 부여된 상태. 그러므로. 2 는 SyntaxError 발생.
  - 사용하지 않는 인자가 전달되어 혼란 야기.

1

```
@router.get("/reservations", ···, dependencies=[Depends(use_user)])
async def reservations(...):
    ...
```

```
@router.get("/reservations", ···)
async def reservations(user: User = Depends(use_user), scheduled_date: datetime.date):
    ...
```

```
@router.get("/reservations", ···)
async def reservations(scheduled_date: datetime.date, user: User = Depends(use_user)):
    ...
```

Query Parameter 값 검사를
HTTP 계층 관심사로 이동

pydantic으로 복잡한 인자를 다루기

> Do Anything with python!

- FastAPI가 데이터를 다루는 데이터 모델 체계를 pydantic으로 다루는 특성을 응용한 기법
- 활용처
  - HTTP Path/Query 인자가 많거나 복잡한 경우
  - 인자값을 검사(validation)하는 방법이 복잡한 경우
  - 인자의 기본값(default)을 동적으로 할당하는 경우

```
class ReservationListParam(pydantic.BaseModel):
    scheduled_date: datetime.date
    search_term: str

    @pydantic.validator("scheduled_date", pre=True, always=True)
    def validate_scheduled_date(cls, v: datetime.date):
        # validate_scheduled_date 함수와 동일
        return v


@router.get("/reservations", response_model=list[schemas.Reservation])
async def reservations(…, params: ReservationListParam = Depends()):
    print(params.scheduled_date)
    print(params.search_term)
    return items
```

FastAPI Depends 색인 노트

> Do Anything with python!

- FastAPI는 컨트롤러가 되도록 Python으로 처리하는 비즈니스 로직 영역에 집중하도록 유도
  - 비즈니스 로직은 Python 함수에게, HTTP 계층은 FastAPI에게, 데이터는 pydantic에게.
- FastAPI가 다루는 HTTP 계층에 대한 맥락은 FastAPI 맥락 영역에서 다루도록 유도
- FastAPI가 관리하는 비동기 작업(async task)에 맞춰 컨트롤러가 호출되는 비동기 작업 맥락을 유지할 때에도 유용
  - 컨트롤러를 호출할 때 마다 호출하는 비동기 맥락 안에서 Depends가 가리키는 의존 대상을 호출.
  - 예를 들어, SQLAlchemy 1.4부터 async를 정식 지원하는데, async scoped session 을 다룰 때 FastAPI의 async task 범위를 벗어나서 문제 발생하는 경우 async scoped session을 가져오는 비동기 함수를 Depends로 호출하여 해결.

결론

학습할 대상의 초점을 좁히기

> Do Anything with python!

- 아는 것이 모르는 것보다 더 친숙하고 익숙, 안도감
  - 사용자 스토리, 시나리오, 보편어 의사 코드, Python 의사 코드, FastAPI 코드
- 빠르고 명확한 피드백 받아 학습 효율 높이는 것이 목적
  - HTTP, 영속 데이터, 비즈니스 로직 등 서로 다른 관심사를 분리
  - 관심사 단위로 구분하여 범위를 명확하게 제한한 테스트 수행
- FastAPI는 그런 점에서 학습하기 좋은 구조

함께 실무에서 활용하고 학습하고 성장해요 > Do Anything with python!

- 성장과 학습 체계 구축과 운용의 방법 중 **일부**
- 페어 프로그래밍, 몹 프로그래밍으로 동료 학습하고 협업하는 방법 등 **Full version과 팀**이 여러분을 기다리고 있습니다.
- **반려동물 의료 데이터** 스타트업인 **(주)아이엠디티**에서 매력있는 기술을 이용해 도전하고 싶은 매력있는 제품을 만들어 **동물 의료 서비스와 새로운 기술이 만나 행복한 반려 라이프를 함께 이룩할** 동료를 모시고 있습니다.
- join@iamdt.co.kr / k.cha@iamdt.co.kr / kay@hannal.net

iamdt™
Integrated Animal Medicine Digital Transformation

PYCON KOREA 2022

감사합니다

> Do Anything with python!