

FastAPI 실습 해설집

기본 설정

```
python

from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
from typing import Optional

app = FastAPI()
```

문제 1 해설 - 사용자 정보 조회

코드

```
python

@app.get("/user/{user_id}")
def get_user(user_id: int):
    return {
        "user_id": user_id,
        "name": f"사용자{user_id}"
    }
```

해설

- `{user_id}`는 주소 매개변수(Path Parameter)로 URL에서 값을 받아옵니다
- FastAPI가 자동으로 문자열을 정수로 변환합니다
- f-string을 사용해 동적으로 사용자 이름을 생성합니다
- 딕셔너리 형태로 응답을 반환하면 FastAPI가 자동으로 JSON으로 변환합니다

실행 결과

```
json

{
  "user_id": 123,
  "name": "사용자123"
}
```

문제 2 해설 - 상품 등록

코드

python

```
class Product(BaseModel):
    name: str
    price: int
    description: str = "설명 없음"

@app.post("/product")
def create_product(product: Product):
    return {
        "message": "상품이 등록되었습니다",
        "product": product
    }
```

해설

- `BaseModel`을 상속받은 Pydantic 모델을 정의합니다
- `description` 필드에 기본값을 설정했습니다
- 함수 매개변수로 Pydantic 모델을 받으면 FastAPI가 자동으로 JSON을 파싱합니다
- 클라이언트가 `description`을 보내지 않으면 기본값이 사용됩니다

실행 결과

json

```
{
  "message": "상품이 등록되었습니다",
  "product": {
    "name": "노트북",
    "price": 1000000,
    "description": "설명 없음"
  }
}
```

문제 3 해설 - 게시글 상세 조회

코드

python

```
@app.get("/post/{post_id}/category/{category}")
def get_post(post_id: int, category: str):
    return {
        "post_id": post_id,
        "category": category,
        "title": f"[{category}] 게시물 {post_id}"
    }
```

해설

- 여러 개의 주소 매개변수를 사용할 수 있습니다
- 함수 매개변수 이름이 URL의 중괄호 안 이름과 일치해야 합니다
- FastAPI가 자동으로 타입 변환과 검증을 수행합니다

실행 결과

```
json

{
  "post_id": 5,
  "category": "notice",
  "title": "[notice] 게시물 5"
}
```

문제 4 해설 - 회원 가입

코드

```
python
```

```

class User(BaseModel):
    email: str
    password: str
    name: str
    age: int = 0
    is_active: bool = True

@app.post("/register")
def register_user(user: User):
    if "@" not in user.email:
        raise HTTPException(status_code=400, detail="잘못된 이메일")

    return {
        "message": "가입 완료",
        "user": user
    }

```

해설

- `Optional`을 사용하지 않고 기본값으로 선택적 필드를 구현했습니다
- `HTTPException`을 사용해 에러를 반환합니다
- `status_code=400`은 잘못된 요청을 나타냅니다
- 간단한 이메일 검증 로직을 추가했습니다

성공 시 실행 결과

```

json
{
  "message": "가입 완료",
  "user": {
    "email": "test@example.com",
    "password": "123456",
    "name": "홍길동",
    "age": 0,
    "is_active": true
  }
}

```

실패 시 실행 결과

```

json

```

```
{  
  "detail": "잘못된 이메일"  
}
```

문제 5 해설 - 주문 상태 업데이트

코드

```
python  
  
class OrderStatus(BaseModel):  
    status: str  
    memo: str = ""  
  
@app.put("/order/{order_id}/status")  
def update_order_status(order_id: int, order_status: OrderStatus):  
    valid_statuses = ["complete", "pending", "cancelled"]  
  
    if order_status.status not in valid_statuses:  
        raise HTTPException(status_code=400, detail="잘못된 상태")  
  
    return {  
        "order_id": order_id,  
        "status": order_status.status,  
        "memo": order_status.memo  
    }
```

해설

- PUT 메서드는 리소스 업데이트에 사용됩니다
- 주소 매개변수와 요청 본문을 동시에 사용할 수 있습니다
- 리스트를 사용해 유효한 상태값을 검증합니다
- `in` 연산자로 값의 존재 여부를 확인합니다

성공 시 실행 결과

```
json  
  
{  
  "order_id": 100,  
  "status": "complete",  
  "memo": "배송 완료"  
}
```

문제 6 해설 - 사용자별 리뷰 작성

코드

python

```
class Review(BaseModel):
    rating: int
    content: str
    recommend: bool = False

@app.post("/user/{user_id}/review/{product_id}")
def create_review(user_id: int, product_id: int, review: Review):
    if review.rating < 1 or review.rating > 5:
        raise HTTPException(status_code=400, detail="평점은 1-5 사이여야 합니다")

    if len(review.content) < 10:
        raise HTTPException(status_code=400, detail="리뷰는 10자 이상 작성해주세요")

    return {
        "user_id": user_id,
        "product_id": product_id,
        "review": review
    }
```

해설

- 복수의 주소 매개변수와 요청 본문을 함께 사용하는 복합적인 엔드포인트입니다
- 여러 조건을 검증하는 복잡한 로직을 구현했습니다
- `len()` 함수로 문자열 길이를 확인합니다
- 논리 연산자(`or`)를 사용해 범위를 검증합니다

성공 시 실행 결과

json

```
{
  "user_id": 1,
  "product_id": 50,
  "review": {
    "rating": 5,
    "content": "정말 좋은 상품이었습니다. 다음에도 구매하고 싶어요!",
    "recommend": true
  }
}
```

전체 코드 (main.py)

```
python
```

```
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
from typing import Optional
```

```
app = FastAPI()
```

```
# 문제 1
```

```
@app.get("/user/{user_id}")
def get_user(user_id: int):
    return {
        "user_id": user_id,
        "name": f"사용자{user_id}"
    }
```

```
# 문제 2
```

```
class Product(BaseModel):
    name: str
    price: int
    description: str = "설명 없음"
```

```
@app.post("/product")
def create_product(product: Product):
    return {
        "message": "상품이 등록되었습니다",
        "product": product
    }
```

```
# 문제 3
```

```
@app.get("/post/{post_id}/category/{category}")
def get_post(post_id: int, category: str):
    return {
        "post_id": post_id,
        "category": category,
        "title": f"[{category}] 게시물 {post_id}"
    }
```

```
# 문제 4
```

```
class User(BaseModel):
    email: str
    password: str
    name: str
    age: int = 0
    is_active: bool = True
```

```
@app.post("/register")
def register_user(user: User):
```



```
if "@" not in user.email:
    raise HTTPException(status_code=400, detail="잘못된 이메일")

return {
    "message": "가입 완료",
    "user": user
}
```

문제 5

```
class OrderStatus(BaseModel):
    status: str
    memo: str = ""

@app.put("/order/{order_id}/status")
def update_order_status(order_id: int, order_status: OrderStatus):
    valid_statuses = ["complete", "pending", "cancelled"]

    if order_status.status not in valid_statuses:
        raise HTTPException(status_code=400, detail="잘못된 상태")

    return {
        "order_id": order_id,
        "status": order_status.status,
        "memo": order_status.memo
    }
```

문제 6

```
class Review(BaseModel):
    rating: int
    content: str
    recommend: bool = False

@app.post("/user/{user_id}/review/{product_id}")
def create_review(user_id: int, product_id: int, review: Review):
    if review.rating < 1 or review.rating > 5:
        raise HTTPException(status_code=400, detail="평점은 1-5 사이여야 합니다")

    if len(review.content) < 10:
        raise HTTPException(status_code=400, detail="리뷰는 10자 이상 작성해주세요")

    return {
        "user_id": user_id,
        "product_id": product_id,
        "review": review
    }
```

실행 방법

1. 위 코드를 `main.py` 파일로 저장
2. 터미널에서 `uvicorn main:app --reload` 실행
3. Thunder Client로 각 엔드포인트 테스트

핵심 개념 정리

주소 매개변수 (Path Parameters)

- URL 경로에 `{변수명}` 형태로 정의
- 함수 매개변수 이름과 일치해야 함
- FastAPI가 자동으로 타입 변환 수행

요청 본문 (Request Body)

- Pydantic 모델로 정의
- POST, PUT 메서드에서 주로 사용
- JSON 형태의 데이터를 자동 파싱

Pydantic 기본값

- `필드명: 타입 = 기본값` 형태로 설정
- 클라이언트가 해당 필드를 보내지 않으면 기본값 사용

에러 처리

- `HTTPException`을 사용해 에러 응답
- `status_code`와 `detail` 메시지 설정 가능