# 11.01 강의

# 공부해볼 것.

## comprehension

## Test isolation

## gather & as_completed 사용법 , 코루틴

## 쉘에서 로그 체크하는 법

poetry add httpx :
Python에서 HTTP 요청을 보내고 응답을 받기 위한 HTTP 클라이언트 라이브러리입니다.

1. **동기 및 비동기 지원**: 비동기 요청을 처리할 수 있어 대규모 네트워크 작업에서 속도와 효율성이 높습니다.
2. **HTTP/2 지원**: 최신 HTTP 표준을 지원하여 빠른 속도의 데이터 전송이 가능합니다.
3. **URL 기반 리다이렉션 관리**: 요청 중 리다이렉션을 자동으로 처리합니다.
4. **타임아웃 및 재시도 설정**: 네트워크의 지연이나 요청 실패 시 자동으로 재시도하거나 시간 제한을 설정할 수 있습니다.

poetry.lock 파일은 반드시 커밋?

diff view 와 edit view (정확한 명칭 아닐 수 있음)

- commit panel 에서 파일을 클릭하면 기본적으로 diff view 에서 열립니다
- diff view 의 왼쪽은 수정 전의 파일, 오른쪽은 수정 후의 파일입니다
- "추가된 라인은" diff view 에서 초록색으로 보입니다.
- "삭제된 라인은" 파란색으로 보입니다.
- diff view 가운데의 '>>' 버튼을 누르면 변경사항을 롤백합니다

- 왼쪽 상단 next difference 버튼으로 수정사항 간에 편리하게 이동할 수 있습니다.

poetry lock 의 hash

- lock 파일이 변경될 때마다 content-hash도 같이 바뀐다.
- content-hash 가 바뀌지 않았다 -> 변경된 종속성이 없다.(아무것도 추가되거나 삭제되지 않았다.)
- content-hash 가 바뀌었다. -> 새로 poetry install을 해줘야 한다.
- github action (CI) 할 때 종속성을 캐싱합니다. (캐싱할 때 hash를 사용합니다)
  -> hash 가 변했을 때에만 캐시를 안쓰고 새로 설치
  -> hash 가 안변했다면 캐시를 사용한다.

```
self.assertEqual(response.status_code, 200)
response_body = response.json()
```

- status code 가 200 임을 먼저 검증했고, 그 다음에 json() 으로 변환한다.
- 이 순서를 반드시 지키는게 좋은데, 안지키면 어떤 문제가 발생할까요?
- http는 json을 주고 받는 protocol 이 아닙니다.
- http는 "문자열"을 주고 받는 protocol 입니다. (TCP 연결을 위해서)
- json 이 아닌 문자열을 json() 으로 변환하려고 시도할 때 decode 에러가 발생합니다.
- 99.9% 의 현대 웹 어플리케이션에서 client 의 요청은 nginx에 먼저 닿습니다.
- client -> Nginx -> uvicorn -> fapsapi
- 만약에 uvicorn 이 다운됐을 경우에 nginx 에서 500 에러를 줍니다.
- 이때 default 500 에러 페이지는 html 입니다. (json이 아닙니다.)
- status _code도 확인하지 않고 json() 호출하면 어떤 에러가 발생하나요?
- JsonDecodeError
- 진짜 문제는 상대방 서버가 다운된게 문제 (500 에러 발생이 진짜 문제)
- 진짜 문제가 JsonDecodeError 에 의해서 숨습니다. (개발자에게 혼란을 준다)

# Round Trip

- 한 번 요청을 하고 그 요청이 들어오기 까지를 Round Trip 이라고 한다.

ORDER BY 가 없는 경우 mysql 의 default 정렬 순서는? -> id 오름차순

테스트 Tip

- 병렬 요청이 가능한 경우 asyncio.gather() 를 사용해서 round trip 횟수를 줄인다.
  -> round trip( 네트워크 round trip )
- http 응답을 json()으로 변환하기 전에 반드시 200 status_code가 왔는지 "먼저" 검증한다.
  -> 500 의 경우에는 반드시 json 으로 응답이 오리란 보장이 없다. (nginx의 default 500 응답은 json이 아니다.)
- 순서가 명시되어 있지 않은 (보장되어 있지 않은) 경우라면 테스트도 순서에 의존하면 안된다.
  -> 순서에 의존하지 않고 검증하는 방법: in 연산 사용하기, 순서 없는 자료구조 사용하기 (set)
  -> dict는 순서가 있을까? -> 파이썬(python 3.6부터)에서 dict는 순서가 있다.
  -> 다른 언어에서는 주로 HashMap Map이라고 부르는데, 보통 순서가 없다.

쉘에서 로그를 확인해야한다. (docker, mysql)
tail -f /var/sqld.log
touch mysqld.log

docker 컨테이너 안에 shell 로 들어가는 방법

1. docker desktop 연다
2. desktop 안의 container 목록 중에서 container 하나를 선택한다.
3. Exec (Execute) 을 선택한다. 선택하면 바로 터미널 창이 보입니다.
4. (내가 root 계정인지 확인하고 whoami ) touch /var/log/mysql.log
    1. touch 명령은 빈 파일을 만든다.

쉘(shell) 종류

1. 쉘에는 여러가지가 있습니다.

2. sh -> 가장 기본적인 쉘
3. bash -> ubuntu 의 기본 쉘
4. zsh 맥의 기본 쉘
5. 내가 지금 sh로 들어와있는데 기본적인 명령어도 찾을 수 없는 경우 bash 명령어를 사용해서 bash로 바꿔 봅니다.
6. bash로 바꿔도 whoami 를 활용해 내가 root 계정인지 체크한다.
7. 명령어 자체를 확인하는 방법->which

파일의 권한 확인

1. ls -al 로 파일의 소유자 (owner) 와 group 을 조회할 수 있다.
2. chown <계정> <파일 이름> 으로 파일의 소유자를 바꿀 수 있다.

docker 안 쓰는 경우 (내가 사용하는 현재 경우다.)

1. 홈 디렉터리에 임의의 파일을 만든다.(mysqld.log 라고 이름 지어도 됩니다)
2.

```
SET global general_log = on;
SET global general_log_file='/Users/dahye/mysqld.log';
SET global log_output = 'file';
```

3. tail -f /Users/{내 계정 이름}/mysqld.log

Test Isolation ***

- Django 와 Tortoise 가 공통적으로 가지고 있는 특징 (TestCase를 사용할 때)
- Test가 시작되면 새로운 데이터베이스를 생성합니다.
- 모든 테스트는 이 새로운 데이터베이스 안에서 이루어집니다.
- 이것을 테스트 공간의 "격리"라고 한다. (격리가 곧 (isolation))
- 격리의 장점 : 기존 데이터에 영향을 받지 않습니다.
    - 기존의 데이터 영향을 받는 경우 : 이미 id가 "comment1" 인 comment 가 존재한다면 테스트시에 id 가 "comment1" 인 comment를 insert 하려고 시도할 때 IntegrityError 가 발생한다. (dup)

- 모든 테스트가 끝나면 데이터 베이스를 삭제합니다.

직렬 await 이 정말로 직렬로 요청하는지 검증하기 (gather가 concurrent 한지 검증)

- article 쿼리가 반환되기 전까지 comment 쿼리가 나가지 않았음을 눈으로 확인했습니다.

asyncio.run( )

- 파이썬 코드가 실행되고 "이벤트 루프"를 실행함으로써 syncio(비동기) 를 쓸 수 있게 됩니다.
- "이벤트 루프"를 실행 시키는 함수가 바로 asyncio.run( ) 입니다.
- asyncio.run( ) 은 이벤트 루프의 실행이 종료될 때 까지 block 합니다.

현재 파일을 실행하는 단축키

- control + shift + r

gather 의 특징

- 시작하자마자 모든 코루틴을 concurrent 하게 실행합니다.
- 가장 마지막 코루틴 까지 완료될 때 까지 block 합니다.
- 인자로 받은 순서대로 그 결과를 반환합니다. (순서가 기억된다.)

as_completed 의 특징

- 시작하자마자 모든 코루틴을 concurrent 하게 실행합니다.
- gather 와 다르게 for 로 반복할 수 있습니다.
- 코루틴이 완료될 때마다 실행 흐름을 돌려 받습니다. (for 내부를 실행 할 수 있습니다.)
- 들어온 순서대로 실행되는 것이 아니라, 코루틴이 완료된 순서대로 실행합니다.
- 먼저 들어온 순서대로 처리해도 되는 경우 gather() 대신에 as_completed 사용가능

변수 이름이 _(언더바) 로 시작하는 경우
캡슐화 를 배웠다면 (privete) (public) 을 알 것이다.

- private : 외부에서 직접 접근하지 말아라
- public
- _ *언더바 둘,* 언더바 하나도 차이가 있습니다.

```
class MyClass:
        def __init__(self, my_var: int) -> None:
                self._my_var = my_var
        def print_my_var(self) -> None:
                """ _my_var 의 주인은? ->
                MyClass MyClass 의 메소드가 _my_var 에 접근한다
                ->      내부에서 접근하는 것 """
                self._private_print_method()
        def _private_print_method(self) -> None:
                print("나는 private")
                print(self._my_var)
```

event loop 의 디테일

- 이전까지 event loop 는 block box 였습니다.
    - 코루틴을 집어 넣으면 알아서 수행이 완료되어서 나오는 마법 상자
- 이제부터는 event loop 를 살짝 들여다 보겠습니다.
- event loop 는 무한 루프인데, 한 번 회전할 때 마다 하는 일이 있습니다.
- 한 번 회전할 때마다, 실행 가능한 모든 코루틴을 scheduled ->
- ready 상태로 이동시키고, ready를 모조리 실행하는게 event loop 의 핵심 동작입니다.