11.04 강의

공부 해야 할 것

제너레이터 (memory efficiency, lazy evaluation)

디버그

제너레이터와 코루틴의 차이

swagger

parsing

스크래치 파일 만드는 단축키 : 커맨드 + 쉬프트 + N
현재 line을 duplicate : 커맨드 + D

제너레이터

- 제너레이터는 "일시 정지가 가능한 함수" 입니다.
- 함수 안에 yield 가 포함되는 순간 함수는 제너레이터로 변합니다.
- 함수는 호출을 하면 그 내부의 코드가 바로 실행되었지만, 제너레이터는 그렇지 않다.
    - 그 대신, 제너레이터 객체가 반환된다.
- 제너레이터 객체를 실행하는 방법
    - for 문을 사용하는 방법
    - 내장 함수 next()를 사용하는 방법
        - next() 는 yield 를 다시 만나기 전까지 제너레이터를 실행한 후에, yield를 만났다면 함수를 다시 일시정지 하고 빠져 나온다.
        - 내장 함수 next()는 제너레이터 객체의 __next__()를 호출한다.
- 제너레이터는 2번 회전(반복) 할 수 없다.
    - 전부 소진된 Iterator 에 다시 한 번 next() 를 호출하면
      (소진되었다 -> 마지막 yield 까지 실행되었다.) StopIteration 에러가

발생합니다.
    - for 문은 반복을 계속 하다가 StopIteration 에러가 발생하면 for 문을 빠져나옵니다.
- generator 는 함수가 실행되고 있는 환경을 그대로 저장한 상태로 함수를 "일시정지" 시킵니다. 따라서 일시정지 할 때 함수의 local 변수도 그대로 저장되어 있으며 다시 함수를 실행할 때 local 변수를 그대로 사용할 수 있습니다.

deque 자료구조란 (double ended queue) -> 끝이 양쪽에 있다.

- double ended -> 끝이 양쪽에 있다
- 원래 queue 는 입구와 출구가 정해져 있다.
- 양쪽 끝을 전부 입구와 출구로 쓸 수 있다
- python 에서 일반적인 queue 를 사용할 때 deque를 쓴다. (heapq 모듈에 필요한 기능이 없는 경우)

deque 룰렛 구현의 불편함

- rullet() 함수 외부에 과거 당첨 금액을 저장해두어야 한다.

debug 중 실행 흐름의 컨트롤

- Resume Program : 다음 중단점 까지 계속 실행
- Step Over : 한줄 실행
- Step Into : 다음 실행하는 함수 내부로 들어간다
- Step Out : 함수 외부로 빠져나온다.

쓰레드

- 쓰레드를 추가하면 실행 흐름이 하나 더 추가됩니다.

함수의 타입

```
my_list = [1,2,3]

def my_func(a:int, b:int, c:int) -> None:
        print(a,b,c)
```

```
my_func(my_list) # 에러 나는 이유는?
# TypeError 발생
# my_func 의 타입 -> int 인자 3개를 받아서 None 을 리턴하는 타입
# 전달할 때 list[int] 하나만 전달했기 때문에
# 인자 타입 mismatch

### my_func(*my_list) 를 작성해야 1,2,3이 모두 들어간다.
# '*' 를 사용하면 리스트를 풀어 헤쳐서 그 안의 모든 요소를 함수에 인자로 전달
# 언패킹이라고 한다.
```

event loop internal

- event loop 내부에는 deque 인 [_ready]와 heap 인 [_scheduled] 가 있다.
- event loop 는 한 번 run 할 때마다 ([_run_once] 라는 함수가 실제로 내부에 있음) 다음에 수행한다.
    - [_scheduled] 에 있는 코루틴 중에서 지금 실행 가능한 코루틴을 [_ready deque]에 밀어 넣는다.
    - [_ready] 안에 모든 코루틴을 순차적으로 실행한다. (랜덤 없음.순차실행.)
    - [_ready] 안에 코루틴을 실행 했다면, 실행 흐름은 이벤트 루프에서 -> 코루틴으로 이동한다.
    - 코루틴 안에서 await을 만나면 (실행 되고 있던 코루틴은 중지되고) 다시 이벤트 루프로 실행 흐름이 이동한다.

미래를 위해 남기는 말

- 여태까지 "코루틴"이란 말을 쓰긴 했는데 엄밀히 말하면 Task 입니다.
    - Task 는 Future의 sub class 입니다.
- 사실은 gather() 를 호출하면 인자로 받은 모든 코루틴은 task 에 감싸집니다.
- 실제로는 await 할 때 task 를 await 할 때만 실행 흐름이 이벤트 루프로 이동합니다. (그냥 일반 코루틴은 이벤트 루프로 실행 흐름이 전환하지 않는다.)

Swagger 문서 자동 생성

- 클라이언트 개발자와 협력할 때에 API Spec 문서는 선택이 아니라 필수입니다.
    - (Python 모르는 개발자에게 서버코드 읽고 실행해보라고 하면 안되겠죠..)
- swagger 문서 접근하는 방법 : FastAPI에서 default 경로는 '/docs' 입니다.
    - http://localhost:8000/docs
- swagger 태그 :
    - api 가 많을 때 tag를 사용해서 분류할 수 있다.
- http method 바꾸기
    - @router. 에서 method를 바꾸면 된다.
- response spec 보여주기 (중요!!!)
- 스펙 문서와 실제 코드의 실행 결과는 "항상" 일치해야 한다.
  - 불일치 한다면, 문서를 읽는 사람이 문서를 불신하게 된다.
  - 이 문제를 최소화 하기 위해서 손으로 문서를 쓰지 않고, 코드로 문서를 생성한다.
- 일치하게 만드는 방법
  1.pydantic 의 BaseModel 을 상속하는 dto class를 만든다.
  2.path operation function 을 등록할 때 response_model 을 전달한다. (1번을 전달한다.)
  3. path operation function 도 해당 dto를 리턴한다.
- 필수 값과 비 필수 값, nullable
  - key가 있을 수도 있고 없을 수도 있다. -> Not Required
  - key 가 무조건 있긴 한데, null 일 수 있다. -> nullable. (강사님 선호)
- Schemas
    - swagger 제일 하단에 모든 spec이 모여있다.

클라이언트 개발자와 협업하는 방법 (진짜 현업에서 하는 방법)
0. (백엔드 프론트 모두 다), 모든 팀원들끼리 상의해서 spec을 먼저 정합니다.

1. DTO와 path operation function 만 먼저 만든다.
2. path operation function 은 dummy dto 를 리턴합니다. dto 안의 값은 임의의 값으로 채우세요.
3. mypy 통과하는지 확인

4. 클라이언트 개발자는 스펙문서를 보고 개발을 시작하고, 백엔드 개발자도 클라이언트가 개발하는 동안 내부의 기능을 채워 넣습니다.

http 기본 구조
http 요청은 다음 요소로 구성된 단순 문자열이다.

- http version (보통 1.1)
- http verb (혹은 method) : (예: GET(조회), POST(생성), PUT(수정), DELETE(삭제), PATCH(수정))
-