Асинхронный контекстный менеджер

Скорее всего, вы уже сталкивались с контекстными менеджерами при работе с файлами:

```
# синхронно открываем фаил для дозасиси в него строки "Hello"
with open('file.txt', 'w+') as fh:
    fh.write("Hello")
```

Асинхронный контеустный менеджер - это объект Python, который обладает магическими методами **`__aenter__`** и **`__aexit__`**. Оба эти метода должны возвращать **`awaitable`** объект, например короутину.

документация

Простенький пример реализации

```
import asyncio
import time
import typing


async def action():
    print(f"{time.time()}: start sleep")
    await asyncio.sleep(1)
    print(f"{time.time()} end sleep")


class AsyncContextManager:
    def __aenter__(self) -> typing.Awaitable:
        print(f"{time.time()}: __aenter__")
        return asyncio.sleep(1)

    async def __aexit__(self, *_) -> typing.Awaitable:
        print(f"{time.time()}: __aexit__")
        return action()


async def main():
    async with AsyncContextManager() as acm:
        pass

    # Добавили сон,чтобы дождаться выполнения action()
    await asyncio.sleep(2)


asyncio.run(main())
```

А вот пример асинхронного запроса в сеть с использованием контекстных менеджеров и библиотеки **aiohttp**, с которой мы более тесно познакомимся на курсе:

```
import asyncio
import aiohttp
from aioresponses import aioresponses # позволяет отловить поход в сеть и вернуть заготовленные данные
#  with aioresponses() as m:
#        m.get(url, payload=dict(crowd=["Andrey", "Alex", "Artem", "Igor"]))


url = "http://service.ru"

async def main():
    with aioresponses() as m:
        m.get(url, payload=dict(crowd=["Andrey", "Alex", "Artem", "Igor"]))
        async with aiohttp.ClientSession() as session:
            async with session.get(url) as response:
                json = await response.json()

    print(json)


data = asyncio.run(main())
```