۱.

توضیح:

می‌خوایم یه کلاس بنویسیم (NestedTimer)که وقتی ازش استفاده می‌کنیم، زمان اجرای یه قسمت از برنامه رو اندازه بگیره.

اگر چند تا تایمر پشت‌سر هم (توی هم) استفاده بشن، هر کدوم با فاصله چاپ بشه تا معلوم بشه کدوم در کدومه.

نکات:

- از متدهای __exit__ و __enter__ .استفاده کنید
- در __enter__ زمان شروع را ذخیره کنید و در
- __exit__ .مدت زمان را حساب کنید
- از یه متغیر کلاس مثلاً (depth_)برای فهمیدن اینکه چند سطح تو در تو هستیم استفاده کنید.

مثال:

```
with NestedTimer("outer"):
    time.sleep(0.1)
    with NestedTimer("inner"):
        time.sleep(0.05)

# خروجی تقریبی:
# [outer] elapsed: 0.100001s
#     [inner] elapsed: 0.050002s
```

۲.

توضیح:

فرض کنید یه برنامه داریم که می‌خواد با چند تا اتصال (connection) همزمان کار کنه (مثل اتصال به دیتابیس یا سرور).
اما تعداد اتصال‌هامون محدوده (مثلاً فقط ۲ تا) .

قراره یه کلاسی به اسم ConnectionPoolبسازید که:

- از دو لیست برای اتصال های آزاد و در حال استفاده استفاده کنید
- وقتی ازش استفاده می‌کنیم(:with pool as conn) ، یکی ازconnection های آزاد رو بهمون می‌ده.
- وقتی از context خارج شدیم، connectionرو آزاد می‌کنه.
- .در صورت پر بودن، خطای NoAvailableConnectionError
- از (__exit__, __enter__)استفاده شود

مثال:

```
pool = ConnectionPool(size=2)

with pool as conn1:
    print("Got", conn1)
    with pool as conn2:
        print("Got", conn2)
```

```
with pool as conn3:  # باید خطا دهد
    pass
```

۳.

- می‌خوایم یه کلاس پایه بسازیم به اسم DataProcessor که یه **قالب کلی برای پردازش داده‌ها** داره.
کلاس‌های دیگه (مثل CSVProcessor و JSONProcessor)از اون ارث‌بری می‌کنن و کلاس‌های فرزند مثل میان و نسخه مخصوص خودشون از این متد(()process_data) رو می‌نویسن.

- کلاس DataProcessorباید **کلاس انتزاعی (abstract)** باشه و متد ()process_dataتوش فقط تعریف بشه ولی بدنه نداشته باشه.
- از teardown و setup برای __exit__ و __enter__ استفاده کنید
- در کلاس‌های فرزند (CSVProcessor, JSONProcessor) متد ()process_dataرو بنویسید.

مثال:

```
with CSVProcessor("data.csv") as p:
    p.run()

with JSONProcessor("data.json") as p:
    p.run()
```

۴.

یه کلاس به اسم UndoRedoManagerبسازید که:

- کارهایی که انجام می‌دیم (مثل "نوشتن" یا "حذف") رو در یه لیست ذخیره کنه.
- وقتی ()undoرو صدا می‌زنیم، آخرین کار رو برگردونه (از لیست حذف‌شده‌ها).
- وقتی ()redoرو صدا می‌زنیم، دوباره همون کار رو انجام بده.
- وقتی از context خارج می‌شیم، وضعیت نهایی رو چاپ کنه.

نکات:

- از دو تا لیست استفاده کنید: یکی برای کارهای انجام‌شده (_done) و یکی برایundone ها.(_undone)
- در ()doهر بار یه کار جدید انجام بدید و لیست undone رو پاک کنید.
- خروجی‌ها باید مشخص و تمیز باشن (مثلاً بنویسه .('[undo] delete 'o

مثال:

```
with UndoRedoManager() as mgr:
    mgr.do("type 'Hello'")
    mgr.do("delete 'o'")
    mgr.undo()
    mgr.redo()
```

۵.

می‌خوایم یه کلاسی بسازید به اسم FunctionProfiler که:

- اگر روی یه تابع قرار گرفت(@FunctionProfiler()) ، زمان اجرای اون تابع رو اندازه بگیره.
- اگر خودش به صورت with استفاده شد، زمان اجرای بلاک رو بگیره.

نکات

- از متدهای __enter__ و __exit__ برای context استفاده کنید.
- از متد __call__ برای دکوراتور استفاده کنید.
- از time.perf_counter() برای اندازه‌گیری زمان استفاده کنید.
- خروجی باید واضح باشه و نشون بده کدوم بخش چند ثانیه طول کشیده.

مثال:

```
@FunctionProfiler("slow_func")
def slow_function():
    time.sleep(0.12)

slow_function()

with FunctionProfiler("block"):
    time.sleep(0.07)
```

۶.

- کلاسی به نام Cache بسازید که به عنوان context manager عمل کند و نتیجه‌ی تابع‌های پرهزینه را ذخیره کند. اگر همان تابع با همان ورودی دوباره فراخوانی شد، نتیجه از cache خوانده شود (یعنی تابع دوباره اجرا نشود)

(خلاصه: می‌خوایم نتایج تابع‌هایی که اجراشون طول می‌کشه رو ذخیره کنیم،
تا وقتی دوباره همون تابع با همون ورودی صدا زده شد، از نتیجه قبلی استفاده بشه.)

نکات:

- از دیکشنری برای ذخیره‌ی نتایج استفاده کنید.
- تابع را با نام و آرگومان‌هایش به عنوان کلید ذخیره کنید.

مثال:

```
:with Cache() as cache

print(cache.run(expensive_func, 10))

print(cache.run(expensive_func, 10))    # بار دوم از cache
```

۷.
کلاسی بنویسید به نام ChainManagerکه بتوان چند متد آن را پشت سر هم صدا زد،
و همه آن‌ها داخل یک context manager مدیریت شوند.

نکات:

- هر متد باید selfبرگرداند تا chain شود.
- در ورود و خروج از context پیام مناسب چاپ شود.
- ساختار کد باید خوانا و مینیمال باشد.

مثال:

```
with ChainManager() as cm:

    cm.step1().step2().step3()
```

۸.
کلاسی بسازید به نام TemporarySettingsکه وقتی وارد context می‌شویم، تنظیمات برنامه را موقتاً تغییر دهد،
و هنگام خروج، همه چیز را به حالت قبل برگرداند.

نکات:

- تنظیمات را در دیکشنری settingsذخیره کنید.
- در __enter__تغییرات اعمال شوند.
- در __exit__همه چیز به مقدار قبلی برگردد.

مثال:

```
settings = {"mode": "normal", "theme": "light"}

with TemporarySettings(settings, mode="debug", theme="dark"):

    print(settings)  # {'mode': 'debug', 'theme': 'dark'}

print(settings)  # {'mode': 'normal', 'theme': 'light'}
```

۹.
کلاسی بنویسید به نام ConfigurableTimerکه هم بتواند با withو هم به عنوان decorator استفاده شود،
و بتوان تعیین کرد که زمان به ثانیه، میلی‌ثانیه یا میکروثانیه نمایش داده شود.

نکات:

- از time.perf_counter استفاده کن.
- یک پارامتر display="s"/"ms"/"µs" برای تنظیم واحد زمان بگذار.
- از clean code استفاده کن و کد تکراری نداشته باش.