1)

کلاسی به نام Fibonacciپیاده کنید که به‌صورت یک ایتراتور (iterator) عمل کند و دنبالهٔ اعداد فیبوناچی را تولید کند.
ایتراتور باید دو شیوهٔ محدودسازی تولید پشتیبانی کند: بر اساس تعداد عناصر (max_count) یا بر اساس مقدار حد بالایی .(max_value) یکی از این دو پارامتر باید یا هر دو اختیاری باشند؛ اگر هیچ‌کدام داده نشود، ValueErrorدهید .

__init__(self, max_count=None, max_value=None):

اگر هر دو None باشند، ValueErrorدهد.

__iter__(self):

آماده‌سازی وضعیت داخلی (ریست) و بازگرداندن خودِ شیء.

__next__(self):

تولید عدد بعدی فیبوناچی. و برای ان شرط توقف بگذارید.

2)

کلاسی به نام Personبسازید که خصوصیات first_name, last_name, birth_yearرا نگه دارد .باید به age صورت property محاسبه شود (براساس سال جاری). برای first_nameو last_name setterاضافه کنید که مقدار ورودی را اعتبارسنجی کند: نام‌ها باید رشته باشند و فقط حروف (یا حداقل شامل حروف و فاصله/نیم فاصلهٔ مجاز در نام‌های مرکب) باشند؛ اگر نام نامعتبر باشد ValueErrorپرتاب شود.

@setter و @property برای

last_name و first_name

که ورودی را تایپ و محتوای رشته را بررسی نمایند و در صورت نامعتبر

.ValueError بدهند

برای توصیفی کوتاه __*str*__ / __repr__

(مثلاً "Person(first_name last_name, birth_year)").

.بدهید ValueError یا TypeError عدد نباشد birth_year اگر

.مجاز است نام‌هایی با فاصله داخلی مانند "مریم احمدی" بپذیرید ولی اعداد یا کاراکتر های عجیب نباید پذیرفته شود

باید همیشه عدد صحیح برگرداند (سال‌ها صحیح هستند) age

3)

یک descriptor به نام BoundedInteger پیاده کنید تا بتوان از آن برای تعریف خصوصیاتی (attributes) استفاده کرد که مقدارشان باید عدد صحیح (int) و در بازهٔ معینی (min..max) باشد. این descriptor باید از طریق __set_name, get, set, delete__پیاده‌سازی شود و پیام‌های خطای مناسب بدهد.

نام باید خصوصی باشد

خطاها بررسی شوند و جلوگیری شوند

4)

کلاسی شبیه contextlib.suppressبسازید که به عنوان context manager عمل کند و یک یا چند نوع استثنا را بگیرد و در صورت وقوع آن‌ها، استثنا را کنترل کند(دیگر بالا نیاید). این کلاس برای تمرین __enter__و __exit__مناسب است و باید رفتار __exit__را به‌درستی کنترل کند.

اگر هیچ استثنایی رخ نداد باید رفتار طبیعی خودش را داشته باشد و برگشت

False/noneداشته باشد

5)

برای فهم عمیق MRO (Method Resolution Order) در پایتون و نحوهٔ استفاده از super()در چند-ارث‌بری، سه کلاس بسازید: یک Logger، یک TimestampMixin، و یک BaseProcessor. سپس کلاسی بسازید که از ترکیب این‌ها ارث می‌برد (مثلاً CombinedProcessor) و در متد مشترک process()از super().process()استفاده کنید تا ترتیب فراخوانی‌ها (chain of responsibility) واضح شود.

class Logger:

با متد process(self) که مثلاً "Logger: logging" را چاپ می‌کند و super().process()را صدا می‌زند یا نزنند (انتخاب را مشخص کنید).

class TimestampMixin:

با متد process(self) که قبل/بعد از super().process() چاپ timestamp یا پیام می‌کند) مثل "Timestamp BEFOREو ("Timestamp AFTER" و سپس super().process().

class BaseProcessor:

با متد process(self) که کار اصلی را انجام می‌دهد) مثلاً."BaseProcessor: processing")

class CombinedProcessor(TimestampMixin, Logger, BaseProcessor):که process() را خودش هم دارد و super().process() را صدا می‌زند تا MRO را نشان دهد.

چاپ CombinedProcessor.*mro* برای نمایش ترتیب جستجوی متدها

6)

یک متاکلاس بسازید که هنگام تعریف کلاس جدید (زمان اجرای class statement) بررسی کند که هیچ‌کدام از attributeها (متغیرهای سطح کلاس) با حرف بزرگ شروع نشده باشند. اگر حداقل یک attribute با حرف بزرگ شروع شده باشد، باید TypeError با پیام واضحی پرتاب شود. هدف تمرین متاکلاس‌ها و درک زمان ساخت کلاس (class creation) است.

متاکلاسی به نام NoMagicUppercaseMeta که type را به ارث ببرد.

در *new* یا *init* متاکلاس، بررسی attrs یا namespace انجام شود: برای هر کلید غیرموجودِ ...که با حرف بزرگ شروع شود، TypeError بدهید.

اجازهٔ تعریف کلاس‌هایی با نام متدهای خاص *dunder* را بدهید (یعنی از بررسی برای

__qualname__ و __module__ صرف نظر کنید)

7)

یک کلاس طراحی کنید که شامل یک فیلد یا شیء غیرقابل pickle (مثلاً یک connection object، file handle، lock، یا شیای که پیکلیبل نیست (باشد. هدف این است که با استفاده از __getstate__ و __setstate__، شی را طوری آمادهٔ pickle کنیم که بخش غیرقابل pickle حذف یا تبدیل شود و پس از pickle.load شی قابل‌بارگذاری و در وضعیت معقولی قرار گیرد (مثلاً آن فیلد برابر None یا بازسازی مجدد شود)

8)

کلاسی Version بسازید که رشته‌های نسخه مانند "1.2.10", "2.0", "1.2.0.1" را نگهداری کند و امکان مقایسهٔ صحیح بین نسخه‌ها را فراهم کند (مثلاً "1.2.10" > "1.2.3"). از functools.total_ordering استفاده کنید تا با پیاده‌سازی فقط __eq__ و __lt__ بقیهٔ مقایسات به‌صورت خودکار ساخته شوند.

9)

یک کلاس ReadOnlyProxy طراحی کنید که یک شیء هدف (target) را می‌پوشاند و به عنوان یک نمایندهٔ خواندنی (read-only) عمل می‌کند: خواندن attribute ها و فراخوانی متدها باید بدون تغییر رفتار به target واگذار شود، اما هر تلاشی برای تغییر (state مثل __setitem__, __setattr__ یا فراخوانی متدهای تغییردهنده‌ای که state را تغییر می‌دهند) باید یا مسدود شود یا سینگال واضحی (استثنا) بدهد. هدف آشنا شدن با delegation ، متدهای خاص (__setitem__, __getitem__, __setattr__, __getattr__) و رفتار proxy است.

10)

طراحی کنید که چندین حساب بانکی را مدیریت کند BankSystem کلاسی به نام.
قابلیت‌ها:

منویی به کاربر نمایش دهد با گزینه‌های زیر:

مشاهده موجودی حساب خاص (با وارد کردن شماره حساب)

برداشت وجه (و نمایش موجودی پس از برداشت)

اضافه کردن حساب جدید

انتقال وجه بین دو حساب

خروج از برنامه

برای اجرای مداوم منو تا انتخاب گزینه‌ی خروج استفاده شود (while True) از ساختار حلقه.

تمام عملیات‌ها باید با متدهای جداگانه مانند show_balance(), withdraw(), add_account(), transfer(), exit_system() پیاده‌سازی شوند.

در صورت خطا (مثل حساب نامعتبر یا موجودی ناکافی)، پیام مناسب نمایش داده شود